Sintaxis y procesamiento de cifrado XML Versión 1.1

Recomendación del W3C 11 de abril de 2013

Esta versión:
http://www.w3.org/TR/2013/REC-xmlenc-core1-20130411/
Última versión publicada:
http://www.w3.org/TR/xmlenc-core1/
Último borrador del editor:
http://www.w3.org/2008/xmlsec/Drafts/xmlenc-core-11/
Versión previa:
http://www.w3.org/TR/2013/PR-xmlenc-core1-20130124/
Editores:
Donald Eastlake , d3e3e3@gmail.com
Joseph Reagle , reagle@mit.edu
Federico Hirsch , frederick.hirsch@nokia.com (1.1)
Thomas Roessler , tlr@w3.org (1.1)
Autores:
Takeshi Imamura , IMAMU@jp.ibm.com
Blair Dillaway , blaird@microsoft.com
Ed Simon , edsimon@xmlsec.com
Kelvin Yiu , kelviny@microsoft.com (1.1)
Magnus Nyström , mnystrom@microsoft.com (1.1)

Consulte las erratas de este documento, que pueden incluir algunas correcciones normativas.

La versión en inglés de esta especificación es la única versión normativa. También pueden estar disponibles traducciones no normativas .

Abstracto

Este documento especifica un proceso para cifrar datos y representar el resultado en XML. Los datos pueden estar en una variedad de formatos, incluidos flujos de octetos y otros datos no estructurados, o formatos de datos estructurados como documentos XML, un elemento XML o contenido de un elemento XML. El resultado del cifrado de datos es un elemento de cifrado XML que contiene o hace referencia a los datos cifrados.

Estado de este documento

Esta sección describe el estado de este documento en el momento de su publicación. Otros documentos pueden reemplazar este documento. Puede encontrar una lista de las publicaciones actuales del W3C y la última revisión de este informe técnico en el índice de informes técnicos del W3C en http://www.w3.org/TR/.

Este documento ha sido revisado por miembros del W3C , desarrolladores de software y otros grupos del W3C y partes interesadas, y cuenta con el respaldo del Director como Recomendación del W3C . Es un documento estable y puede usarse como material de referencia o citarse de otro documento. El papel del W3C al elaborar la Recomendación es llamar la atención sobre la especificación y promover su implementación generalizada. Esto mejora la funcionalidad y la interoperabilidad de la Web.

La versión original de esta especificación fue producida por el Grupo de Trabajo de Cifrado XML del W3C ; El Informe de interoperabilidad muestra cuatro implementaciones con al menos dos implementaciones interoperables en cada característica.

Consulte el informe de implementación de la versión 1.1 de esta especificación para obtener detalles adicionales sobre el estado de implementación de las funciones agregadas en esta revisión.

Los cambios que afectan la conformidad con respecto a la Recomendación anterior afectan principalmente al conjunto de algoritmos criptográficos obligatorios para implementar, al agregar el Acuerdo de claves Diffie-Hellman de curva elíptica, hacer obligatorio AES-128 GCM, cambiar RSA v1.5 a opcional, agregar AES192-GCM opcional y agregando variantes opcionales del algoritmo RSA-OEAP. También se han agregado importantes consideraciones de seguridad. Un resumen detallado de los cambios está disponible en [XMLENC-CORE1-CHGS]. Los cambios también se describen en un documento de diferencias que muestra los cambios desde la Recomendación original , así como en un documento de diferencias que muestra los cambios desde el borrador de PR anterior .

Este documento fue publicado por el Grupo de Trabajo de Seguridad XML como recomendación. Si desea hacer comentarios sobre este documento, envíelos a public-xmlsec@w3.org (suscríbase , archivos). Todos los comentarios son bienvenidos.

Este documento fue elaborado por un grupo que opera bajo la Política de Patentes del W3C del 5 de febrero de 2004 . El W3C mantiene una lista pública de cualquier divulgación de patentes realizada en relación con los productos del grupo; esa página también incluye instrucciones para divulgar una patente. Una persona que tenga conocimiento real de una patente que cree que contiene Reivindicaciones Esenciales debe revelar la información de acuerdo con la sección 6 de la Política de Patentes del W3C .

También está disponible información adicional relacionada con el estado de los derechos de propiedad intelectual de XML Encryption 1.1 .

Tabla de contenido

1. Introducción

Este documento especifica un proceso para cifrar datos y representar el resultado en XML. Los datos pueden ser datos arbitrarios (incluido un documento XML), un elemento XML o contenido de un elemento XML. El resultado del cifrado de datos es un elemento de cifrado XML `EncryptedData`que contiene (a través de uno de sus contenidos secundarios) o identifica (a través de una referencia URI) los datos cifrados.

Al cifrar un elemento XML o el contenido del elemento, el `EncryptedData`elemento reemplaza el elemento o el contenido (respectivamente) en la versión cifrada del documento XML.

Al cifrar datos arbitrarios (incluidos documentos XML completos), el `EncryptedData`elemento puede convertirse en la raíz de un nuevo documento XML o convertirse en un elemento secundario en un documento XML elegido por la aplicación.

1.1 Convenciones editoriales y de conformidad

Esta especificación utiliza esquemas XML [XMLSCHEMA-1], [XMLSCHEMA-2] para describir el modelo de contenido. La gramática normativa completa está definida por el esquema XSD y el texto normativo de esta especificación. El archivo de esquema XSD independiente tiene autoridad en caso de que exista algún desacuerdo entre él y las partes del esquema XSD.

Las palabras clave " DEBE ", " NO DEBE ", " REQUERIDO ", " DEBE ", " NO DEBE " , " DEBE ", " NO DEBE ", " RECOMENDADO ", " PUEDE " y " OPCIONAL " en esta especificación son debe interpretarse como se describe en [RFC2119]:

> " Sólo DEBEN utilizarse cuando sea realmente necesario para la interoperación o para limitar comportamientos que puedan causar daño (por ejemplo, limitar las retransmisiones)"

Consequently, we use these capitalized keywords to unambiguously specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations. These key words are not used (capitalized) to describe XML grammar; schema definitions unambiguously describe such requirements and we wish to reserve the prominence of these terms for the natural language descriptions of protocols and features. For instance, an XML attribute might be described as being "optional". Compliance with the XML-namespace specification [XML-NAMES] is described as "REQUIRED".

1.2 Design Philosophy

The design philosophy and requirements of this specification (including the limitations related to instance validity) are addressed in the original XML Encryption Requirements [XML-ENCRYPTION-REQ] and the XML Security 1.1 Requirements document [XMLSEC11-REQS].

1.3 Versions, Namespaces, URIs, and Identifiers

This specification makes use of XML namespaces, and uses Uniform Resource Identifiers [URI] to identify resources, algorithms, and semantics.

Implementations of this specification MUST use the following XML namespace URIs:

URI	namespace prefix	XML internal entity
`http://www.w3.org/2001/04/xmlenc#`	*default namespace*, `xenc:`	`<!ENTITY xenc "http://www.w3.org/2001/04/xmlenc#">`
`http://www.w3.org/2009/xmlenc11#`	`xenc11:`	`<!ENTITY xenc11 "http://www.w3.org/2009/xmlenc11#">`

The `http://www.w3.org/2001/04/xmlenc#` (`xenc:`) namespace was introduced in version 1.0 of this specification. The present version does not coin any new elements or algorithm identifiers in that namespace; instead, the `http://www.w3.org/2009/xmlenc11#` (`xenc11:`) namespace is used.

No provision is made for an explicit version number in this syntax. If a future version of this specification requires explicit versioning of the document format, a different namespace will be used.

Additionally, this specification uses elements and algorithm identifiers from the XML Signature name spaces [XMLDSIG-CORE1]:

URI	namespace prefix	XML internal entity
`http://www.w3.org/2000/09/xmldsig#`	*default namespace*, `ds:`, `dsig:`	`<!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">`
`http://www.w3.org/2009/xmldsig11#`	`dsig11:`	`<!ENTITY dsig11 "http://www.w3.org/2009/xmldsig11#">`

1.4 Acknowledgements

Se agradecen las contribuciones de los siguientes miembros del Grupo de Trabajo original a la especificación de cifrado XML original de acuerdo con las [políticas de contribuyentes y la lista]() activa del Grupo de Trabajo : Joseph Ashwood, Simon Blake-Wilson, Certicom, Frank D. Cavallito, BEA Systems, Eric Cohen, PricewaterhouseCoopers, Blair Dillaway, Microsoft (Autor), Blake Dournaee, RSA Security, Donald Eastlake, Motorola (Editor), Barb Fox, Microsoft, Christian Geuer-Pollmann, Universidad de Siegen, Tom Gindin, IBM, Jiandong Guo, Phaos , Phillip Hallam-Baker, Verisign, Amir Herzberg, NewGenPay, Merlin Hughes, Baltimore, Frederick Hirsch, Maryann Hondo, IBM, Takeshi Imamura, IBM (Autor), Mike Just, Entrust, Inc., Brian LaMacchia, Microsoft, Hiroshi Maruyama, IBM, John Messing, Law-on-Line, Shivaram Mysore, Sun Microsystems, Thane Plambeck, Verisign, Joseph Reagle, W3C (Presidente, Editor), Aleksey Sanin, Jim Schaad, Soaring Hawk Consulting, Ed Simon, XMLsec (Autor), Daniel Toth, Ford, Yongge Wang, Certicom, Steve Wiley, myProof.

Además, agradecemos a las siguientes personas por sus comentarios durante y después de la última convocatoria de la recomendación original: Martin Dürst, W3C , Dan Lanz, Zolera, Susan Lesch, W3C , David Orchard, BEA Systems, Ronald Rivest, MIT .

Se recibieron contribuciones para la versión 1.1 de los miembros del Grupo de Trabajo de Seguridad XML: Scott Cantor, Juan Carlos Cruellas, Pratik Datta, Gerald Edgar, Ken Graf, Phillip Hallam-Baker, Brad Hill, Frederick Hirsch, Brian LaMacchia, Konrad Lanz, Hal Lockhart, Cynthia Martin, Rob Miller, Sean Mullan, Shivaram Mysore, Magnus Nyström, Bruce Rich, Thomas Roessler, Ed Simon, Chris Solc, John Wray, Kelvin Yiu.

El grupo de trabajo también reconoce la contribución de Juraj Somorovsky al plantear la cuestión del ataque de texto cifrado elegido por CBC y las contribuciones a la revisión de las consideraciones de seguridad de XML Encryption 1.1.

2. Descripción general y ejemplos de cifrado

Esta sección no es normativa.

Esta sección proporciona una descripción general y ejemplos de la sintaxis de cifrado XML. La sintaxis formal se encuentra en [la sección 3. Sintaxis de cifrado]() ; el procesamiento específico se proporciona en [las Reglas de procesamiento]() (sección 4).

Expresado en forma abreviada, el `EncryptedData`elemento tiene la siguiente estructura (donde "?" denota cero o una ocurrencia; "+" denota una o más ocurrencias; "*" denota cero o más ocurrencias; "|" denota una elección; y el elemento vacío etiqueta de elemento significa que el elemento debe estar vacío):

EJEMPLO 1

```
< Identificación de datos cifrados ? ¿Tipo ? ¿ Tipo Mime ? ¿ Codificación ? > <Método de cifrado/> ?
  <ds:KeyInfo> <Clave cifrada> ?
    <Método de acuerdo> ?
    <ds:NombreClave> ?
    <ds:MétododeRecuperación> ?
    <ds :* > ?
  </ds:KeyInfo> ?
  <DatosCifrado> <ValorCifrado> | < URI de referencia de cifrado ? > </CipherData> <EncryptionProperties> ?
</EncryptedData>
```

El `CipherData`elemento envuelve o hace referencia a los datos cifrados sin procesar. Un `CipherData`elemento debe tener un elemento secundario `CipherValue`o `CipherReference`. Si es envolvente, los datos cifrados sin procesar son el `CipherValue`contenido del elemento; si se hace referencia, el atributo `CipherReference`del elemento `URI`apunta a la ubicación de los datos cifrados sin procesar

2.1 Granularidad del cifrado

Esta sección no es normativa.

Nota: Los ejemplos de este documento no consideran ataques de adivinación de texto sin formato ni otros riesgos, y solo tienen fines ilustrativos.

Considere la siguiente información de pago ficticia, que incluye información de identificación e información apropiada para un método de pago (por ejemplo, tarjeta de crédito, transferencia de dinero o cheque electrónico):

EJEMPLO 2

```
<? versión xml = "1.0" ?>

<PaymentInfo xmlns = "http://example.org/paidv2" > <Nombre> John Smith </Nombre> < Límite de tarjeta de crédito = "5000" Mor
```

Este margen representa que John Smith está usando su tarjeta de crédito con un límite de $5,000USD.

2.1.1 Cifrar un elemento XML

Esta sección no es normativa.

¡El número de tarjeta de crédito de Smith es información confidencial! Si la aplicación desea mantener esa información confidencial, puede cifrar el `CreditCard`elemento:

EJEMPLO 3

```
<? versión xml = "1.0" ?>

<PaymentInfo xmlns = "http://example.org/paidv2" > <Nombre> John Smith </Nombre> <EncryptedData Type = "http://www.w3.org/2C
```

Al cifrar todo el `CreditCard`elemento desde sus etiquetas de inicio a fin, se oculta la identidad del elemento en sí. (Un espía no sabe si utilizó una tarjeta de crédito o una transferencia de dinero). El `CipherData`elemento contiene la serialización cifrada del `CreditCard`elemento.

2.1.2 Cifrado del contenido del elemento XML (Elementos)

Como escenario alternativo, puede ser útil para los agentes intermediarios saber que John usó una tarjeta de crédito con un límite particular, pero no el número, el emisor y la fecha de vencimiento de la tarjeta. En este caso, el contenido (datos de caracteres o elementos secundarios) del `CreditCard`elemento se puede cifrar:

EJEMPLO 4

```
<? versión xml = "1.0" ?>

<PaymentInfo xmlns = "http://example.org/paidv2" > <Nombre> John Smith </Nombre> < Límite de tarjeta de crédito = "5000" Mor
```

2.1.3 Cifrado del contenido del elemento XML (datos de caracteres)

Alternativamente, considere el escenario en el que toda la información, *excepto* el número de tarjeta de crédito real, puede estar clara, incluido el hecho de que existe el elemento Número:

EJEMPLO 5

```
<? versión xml = "1.0" ?>

<PaymentInfo xmlns = "http://example.org/paidv2" > <Nombre> John Smith </Nombre> < Límite de tarjeta de crédito = "5000" Mor
```

Ambos `CreditCard`y `Number`están claros, pero el contenido de los datos de caracteres `Number`está cifrado.

2.1.4 Cifrado de datos arbitrarios y documentos XML

Si el escenario de la aplicación requiere que toda la información esté cifrada, todo el documento se cifra como una secuencia de octetos. Esto se aplica a datos arbitrarios, incluidos documentos XML.

EJEMPLO 6

```
<? versión xml = "1.0" ?>

<EncryptedData xmlns = "http://www.w3.org/2001/04/xmlenc#" MimeType = "text/xml" > <CipherData> <CipherValue> A23B45C56 </Ci
```

Cuando corresponda, como en el caso de cifrar un flujo EXI completo, SE DEBE proporcionar el atributo Tipo e indicar el uso de EXI. El MimeType opcional PUEDE usarse para registrar el tipo real (no codificado EXI), pero no es necesario y puede omitirse, como en el siguiente ejemplo de cifrado EXI:

EJEMPLO 7

```
<? versión xml = "1.0" ?>

<EncryptedData xmlns = "http://www.w3.org/2001/04/xmlenc#" Tipo = "http://www.w3.org/2009/xmlenc11#EXI" > <CipherData> <Ciph
```

2.1.5 Supercifrado: cifrado de datos cifrados

Un documento XML puede contener cero o más `EncryptedData` elementos. `EncryptedData`no puede ser padre o hijo de otro `EncryptedData`elemento. Sin embargo, los datos reales cifrados pueden ser cualquier cosa, incluidos `EncryptedData`elementos `EncryptedKey` (es decir, supercifrado). Durante el supercifrado de un elemento `EncryptedData`o `EncryptedKey`, se debe cifrar todo el elemento. Cifrar solo el contenido de estos elementos o cifrar elementos secundarios seleccionados es una instancia no válida según el esquema proporcionado.

Por ejemplo, considere lo siguiente:

EJEMPLO 8

```
<pay:PaymentInfo xmlns:pay = "http://example.org/paidv2" > <EncryptedData Id = "ED1" xmlns = "http://www.w3.org/2001/04/xmle
```

Un supercifrado válido de " `//xenc:EncryptedData[@Id='ED1']`" sería:

EJEMPLO 9

```
<pay:PaymentInfo xmlns:pay = "http://example.org/paidv2" > <EncryptedData Id = "ED2" xmlns = "http://www.w3.org/2001/04/xmle
```

donde el `CipherValue`contenido de ' `newEncryptedData`' es la codificación base64 de la secuencia de octetos cifrados resultante del cifrado del `EncryptedData`elemento con `Id='ED1'`.

2.2`EncryptedData` y `EncryptedKey`uso

2.2.1`EncryptedData` con clave simétrica (`KeyName`)

EJEMPLO 10

```
[ s01 ]< EncryptedData xmlns = "http://www.w3.org/2001/04/xmlenc#" Tipo = "http://www.w3.org/2001/04/xmlenc#Element" > [ s02



[s05] </ ds : KeyInfo > [ s06 ] < CipherData >< CipherValue > DEADBEEF </CipherValue> < / CipherData > [ s07 ]</ EncryptedDa
```

`[s1]`El tipo de datos cifrados se puede representar como un valor de atributo para ayudar en el descifrado y el procesamiento posterior. En este caso, los datos cifrados eran un "elemento". Otras alternativas incluyen el "contenido" de un elemento o una secuencia de octetos externa que también puede identificarse mediante los atributos `MimeType`y `Encoding`.

`[s2]`Este (3DES CBC) es un cifrado de clave simétrica.

`[s4]`La clave simétrica tiene un nombre asociado "John Smith".

`[s6]` `CipherData`contiene un `CipherValue`, que es una secuencia de octetos codificada en base64. Alternativamente, podría contener un `CipherReference`, que es una referencia de URI junto con las transformaciones necesarias para obtener los datos cifrados como una secuencia de octetos.

2.2.2`EncryptedKey` (`ReferenceList`, `ds:RetrievalMethod`, `CarriedKeyName`)

La siguiente `EncryptedData`estructura es muy similar a la anterior, excepto que esta vez se hace referencia a la clave mediante `ds:RetrievalMethod`:

EJEMPLO 11

```
[ t01 ]< Id. de datos cifrados = "ED"
                    xmlns = "http://www.w3.org/2001/04/xmlenc#" > [ t02 ] < Algoritmo del método de cifrado = "http://www.w3



[t06] </ ds : KeyInfo > [ t07 ] < CipherData >< CipherValue > DEADBEEF </CipherValue> < / CipherData > [ t08 ]</ EncryptedDa
```

`[t02]`Este (AES-128-CBC) es un cifrado de clave simétrica.

[t04] ds:RetrievalMethodse utiliza para indicar la ubicación de una clave con tipo xenc:EncryptedKey. La clave (AES) se encuentra en '#EK'.

[t05] ds:KeyNameproporciona un método alternativo para identificar la clave necesaria para descifrar el archivo CipherData. Cualquiera o ambos ds:KeyNamey ds:KeyRetrievalMethod podrían usarse para identificar la misma clave.

Dentro del mismo documento XML, existía una EncryptedKey estructura a la que se hacía referencia en [t04]:

EJEMPLO 12

```
[ t09 ]< Id. de clave cifrada = "EK" xmlns = "http://www.w3.org/2001/04/xmlenc#" > [ t10 ] < Algoritmo del método de cifradc

[t13] </ ds : KeyInfo > [ t14 ] < CipherData >< CipherValue > xyzabc </CipherValue> < / CipherData > [ t15 ] < Lista de refe

[t18] <CarriedKeyName>Sally Doe</ CarriedKeyName > [ t19 ]</ EncryptedKey >
```

[t09]El EncryptedKeyelemento es similar al EncryptedDataelemento excepto que los datos cifrados son siempre un valor clave.

[t10]Es EncryptionMethodel algoritmo de clave pública RSA.

[t12] ds:KeyNamede "John Smith" es una propiedad de la clave necesaria para descifrar (usando RSA) el archivo CipherData.

[t14]El CipherData's CipherValue es una secuencia de octetos que es procesada (serializada, cifrada y codificada) por un objeto cifrado de referencia EncryptionMethod. (Tenga en cuenta que una EncryptedKey EncryptionMethodes el algoritmo utilizado para cifrar estos octetos y no habla de qué tipo de octetos son).

[t15-17]A ReferenceListidentifica los objetos cifrados (DataReferencey KeyReference) cifrados con esta clave. Contiene ReferenceListuna lista de referencias a datos cifrados por la clave simétrica contenida dentro de esta estructura.

[t18]El CarriedKeyNameelemento se utiliza para identificar el valor de la clave cifrada al que puede hacer referencia el KeyNameelemento en ds:KeyInfo. (Dado que los valores de los atributos de ID deben ser exclusivos de un documento, CarriedKeyNamepuede indicar que varias EncryptedKeyestructuras contienen el mismo valor de clave cifrado para diferentes destinatarios).

3. Sintaxis de cifrado

Esta sección proporciona una descripción detallada de la sintaxis y las funciones del cifrado XML. Las características descritas en esta sección DEBEN implementarse a menos que se indique lo contrario. La sintaxis se define mediante [XMLSCHEMA-1], [XMLSCHEMA-2] con el siguiente preámbulo XML, declaración, entidad interna e importación:

```
Definición del esquema :

<? versión xml = "1.0" codificación = "utf-8" ?> <!DOCTYPE esquema PUBLIC "-//W3C//DTD XMLSchema 200102//EN"

[
<!ATTLIST esquema
xmlns:xenc CDATA #FIXED 'http://www.w3.org/2001/04/xmlenc'#
xmlns:ds CDATA #FIXED 'http://www.w3.org/2000/09/xmldsig#'> <!ENTITY xenc 'http://www.w3.org/2001/04/xmlenc#'> <! ENTIDAD % p

]>

<esquema xmlns = "http://www.w3.org/2001/XMLSchema" versión = "1.0" xmlns:ds = "http://www.w3.org/2000/09/xmldsig#" xmlns:xenc

  <importar espacio de nombres = "http://www.w3.org/2000/09/xmldsig#" esquemaLocation = "http://www.w3.org/TR/2002/
            REC-xmldsig-core-20020212/xmldsig-core-schema.xsd" />
```

(Nota: se agregó una nueva línea al URI de ubicación de esquema para que quepa en esta página, pero no forma parte del URI).

El marcado adicional definido en esta especificación utiliza el xenc11:espacio de nombres. La sintaxis se define en un esquema XML con el siguiente preámbulo:

```
Definición del esquema :

<? versión xml = "1.0" codificación = "utf-8" ?> <!DOCTYPE esquema PUBLIC "-//W3C//DTD XMLSchema 200102//EN"

[
<!ATTLIST esquema
xmlns:xenc CDATA #FIXED "http://www.w3.org/2001/04/xmlenc#"
xmlns:ds CDATA #FIXED "http://www.w3.org/2000/09/xmldsig#"
xmlns:xenc11 CDATA #FIXED "http://www.w3.org/2009/xmlenc11#"> <!ENTITY xenc "http://www.w3.org/2001/04/xmlenc#"> <!ENTITY % p

]>

<esquema xmlns = "http://www.w3.org/2001/XMLSchema" versión = "1.0" xmlns:xenc = "http://www.w3.org/2001/04/xmlenc#" xmlns:xer

  <importar espacio de nombres = "http://www.w3.org/2000/09/xmldsig#" esquemaLocation = "http://www.w3.org/TR/2002/
```

```
                    REC-xmldsig-core-20020212/xmldsig-core-schema.xsd" />

   <importar espacio de nombres = "http://www.w3.org/2001/04/xmlenc#" esquemaLocation = "http://www.w3.org/TR/2002/

                    REC-xmlenc-core-20021210/xenc-schema.xsd" />
```

(Nota: se agregó una nueva línea al URI de ubicación de esquema para que quepa en esta página, pero no forma parte del URI).

3.1 El EncryptedTypeelemento

EncryptedTypees el tipo abstracto del que se derivan EncryptedDatay . EncryptedKeySi bien estos dos últimos tipos de elementos son muy similares con respecto a sus modelos de contenido, una distinción sintáctica es útil para el procesamiento. Las implementaciones DEBEN generar un esquema laxamente válido [XMLSCHEMA-1], [XMLSCHEMA-2] EncryptedDatao EncryptedKeyelementos según lo especificado en las declaraciones de esquema posteriores. (Tenga en cuenta que la generación válida del esquema laxo significa que el contenido permitido xsd:ANYno necesita ser válido). Las implementaciones DEBEN crear estas estructuras XML (EncryptedTypeelementos y sus descendientes/contenido) en el Formulario de normalización C [NFC].

```
Definición del esquema :

<complexType name="EncryptedType" abstract="true">
  <sequence>
    <element name="EncryptionMethod" type="xenc:EncryptionMethodType"
             minOccurs="0"/>
    <element ref="ds:KeyInfo" minOccurs="0"/>
    <element ref="xenc:CipherData"/>
    <element ref="xenc:EncryptionProperties" minOccurs="0"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
  <attribute name="Type" type="anyURI" use="optional"/>
  <attribute name="MimeType" type="string" use="optional"/>
  <attribute name="Encoding" type="anyURI" use="optional"/>
</complexType>
```

EncryptionMethod is an optional element that describes the encryption algorithm applied to the cipher data. If the element is absent, the encryption algorithm must be known by the recipient or the decryption will fail.

ds:KeyInfo is an optional element, defined by [XMLDSIG-CORE1], that carries information about the key used to encrypt the data. Subsequent sections of this specification define new elements that may appear as children of ds:KeyInfo.

CipherData is a mandatory element that contains the CipherValue or CipherReference with the encrypted data.

EncryptionProperties can contain additional information concerning the generation of the EncryptedType (e.g., date/time stamp).

Id is an optional attribute providing for the standard method of assigning a string id to the element within the document context.

Type is an optional attribute identifying type information about the plaintext form of the encrypted content. While optional, this specification takes advantage of it for processing described in section 4.4 Decryption. If the EncryptedData element contains data of Type 'element' or element 'content', and replaces that data in an XML document context, or contains data of Type 'EXI', it is strongly recommended the Type attribute be provided. Without this information, the decryptor will be unable to automatically restore the XML document to its original cleartext form.

MimeType is an optional (advisory) attribute which describes the media type of the data which has been encrypted. The value of this attribute is a string with values defined by [RFC2045]. For example, if the data that is encrypted is a base64 encoded PNG, the transfer Encoding may be specified as 'http://www.w3.org/2000/09/xmldsig#base64' and the MimeType as 'image/png'. This attribute is purely advisory; no validation of the MimeType information is required and it does not indicate the encryption application must do any additional processing. Note, this information may not be necessary if it is already bound to the identifier in the Type attribute. For example, the Element and Content types defined in this specification are always UTF-8 encoded text. In the case of Type EXI the MimeType attribute is not necessary, but if used should reflect the underlying type and not "EXI".

Encoding is an optional (advisory) attribute which describes the transfer encoding of the data that has been encrypted.

3.2 The EncryptionMethod Element

EncryptionMethod is an optional element that describes the encryption algorithm applied to the cipher data. If the element is absent, the encryption algorithm must be known to the recipient or the decryption will fail.

```
Schema Definition:

<complexType name="Encryptionmethodtype" mixed="true">
  <sequence>
    <element name="KeySize" minOccurs="0" type="xenc:KeySizeType"/>
    <element name="OAEPparams" minOccurs="0" type="base64Binary"/>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Algorithm" type="anyURI" use="required"/>
</complexType>
```

The permitted child elements of the EncryptionMethod are determined by the specific value of the Algorithm attribute URI, and the KeySize child element is always permitted. For example, the RSA-OAEP algorithm (section 5.5.2 RSA-OAEP) uses the ds:DigestMethod and OAEPparams elements, and may use the xenc11:MGF element when needed. (We rely upon the ANY schema construct because it is not possible to specify element content based on the value of an attribute.)

The presence of any child element under EncryptionMethod that is not permitted by the algorithm or the presence of a KeySize child inconsistent with the algorithm MUST be treated as an error. (All algorithm URIs specified in this document imply a key size but this is not true in general. Most popular stream cipher algorithms take variable size keys.)

3.3 The CipherData Element

The CipherData is a mandatory element that provides the encrypted data. It must either contain the encrypted octet sequence as base64 encoded text as element content of the CipherValue element, or provide a reference to an external location containing the encrypted octet sequence via the CipherReference element.

Schema Definition:

```
<element name="CipherData" type="xenc:CipherDataType"/>

<complexType name="CipherDataType">
  <choice>
    <element name="CipherValue" type="base64Binary"/>
    <element ref="xenc:CipherReference"/>
  </choice>
</complexType>
```

3.3.1 The `CipherReference` Element

If `CipherValue` is not supplied directly, the `CipherReference` identifies a source which, when processed, yields the encrypted octet sequence.

The actual value is obtained as follows. The `CipherReference URI` contains an identifier that is dereferenced. Should the `CipherReference` element contain an OPTIONAL sequence of `Transform`s, the data resulting from dereferencing the URI is transformed as specified so as to yield the intended cipher value. For example, if the value is base64 encoded within an XML document; the transforms could specify an XPath expression followed by a base64 decoding so as to extract the octets.

The syntax of the URI and Transforms is defined in XML Signature [XMLDSIG-CORE1], however XML Encryption places the `Transforms` element in the XML Encryption namespace since it is used in XML Encryption to obtain an octet stream for decryption. In [XMLDSIG-CORE1] both generation and validation processing start with the same source data and perform that transform in the same order. In encryption, the decryptor has only the cipher data and the specified transforms are enumerated for the decryptor, in the order necessary to obtain the octets. Consequently, because it has different semantics `Transforms` is in the `xenc:` namespace.

For example, if the relevant cipher value is captured within a `CipherValue` element within a different XML document, the `CipherReference` might look as follows:

EXAMPLE 13

```
<CipherReference URI="http://www.example.com/CipherValues.xml">
  <Transforms>
    <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
      <ds:XPath xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
        self::text()[parent::enc:CipherValue[@Id="example1"]]
      </ds:XPath>
    </ds:Transform>
    <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#base64"/>
  </Transforms>
</CipherReference>
```

Implementations MUST support the `CipherReference` feature and the same URI encoding, dereferencing, scheme, and HTTP response codes as that of [XMLDSIG-CORE1]. The `Transform` feature and particular transform algorithms are OPTIONAL.

Schema Definition:

```
<element name="CipherReference" type="xenc:CipherReferenceType"/>

<complexType name="CipherReferenceType">
  <sequence>
    <element name="Transforms" type="xenc:TransformsType" minOccurs="0"/>
  </sequence>
  <attribute name="URI" type="anyURI" use="required"/>
</complexType>

<complexType name="TransformsType">
  <sequence>
    <element ref="ds:Transform" maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

3.4 The `EncryptedData` Element

The `EncryptedData` element is the core element in the syntax. Not only does its `CipherData` child contain the encrypted data, but it's also the element that replaces the encrypted element, or element content, or serves as the new document root.

Schema Definition:

```
<element name="EncryptedData" type="xenc:EncryptedDataType"/>

<complexType name="EncryptedDataType">
  <complexContent>
    <extension base="xenc:EncryptedType">
    </extension>
  </complexContent>
</complexType>
```

3.5 Extensions to `ds:KeyInfo` Element

There are three ways that the keying material needed to decrypt `CipherData` can be provided:

1. The `EncryptedData` or `EncryptedKey` element specify the associated keying material via a child of `ds:KeyInfo`. All of the child elements of ds:`KeyInfo` specified in [XMLDSIG-CORE1] MAY be used as qualified:
   1. Support for `ds:KeyValue` is OPTIONAL and may be used to transport public keys, such as Diffie-Hellman Key Values (section 5.6.1 Diffie-Hellman Key Values). (Including the plaintext decryption key, whether a private key or a secret key, is obviously NOT RECOMMENDED.)
   2. Support of `ds:KeyName` to refer to an `EncryptedKey` `CarriedKeyName` is RECOMMENDED.
   3. Support for same document `ds:RetrievalMethod` is REQUIRED.

   In addition, we provide two additional child elements: applications MUST support `EncryptedKey` (section 3.5.1 The EncryptedKey Element) and MAY support `AgreementMethod` (section 5.6 Key Agreement).

2. A detached (not inside `ds:KeyInfo`) `EncryptedKey` element can specify the `EncryptedData` or `EncryptedKey` to which its decrypted key will apply via a `DataReference` or `KeyReference` (section 3.6 The ReferenceList Element).
3. The keying material can be determined by the recipient by application context and thus need not be explicitly mentioned in the transmitted XML.

3.5.1 The `EncryptedKey` Element

Identifier

`Type="http://www.w3.org/2001/04/xmlenc#EncryptedKey"`

(This can be used within a `ds:RetrievalMethod` element to identify the referent's type.)

The `EncryptedKey` element is used to transport encryption keys from the originator to a known recipient(s). It may be used as a stand-alone XML document, be placed within an application document, or appear inside an `EncryptedData` element as a child of a `ds:KeyInfo` element. The key value is always encrypted to the recipient(s). When `EncryptedKey` is decrypted the resulting octets are made available to the `EncryptionMethod` algorithm without any additional processing.

```
Schema Definition:

<element name="EncryptedKey" type="xenc:EncryptedKeyType"/>

<complexType name="EncryptedKeyType">
  <complexContent>
    <extension base="xenc:EncryptedType">
      <sequence>
        <element ref="xenc:ReferenceList" minOccurs="0"/>
        <element name="CarriedKeyName" type="string" minOccurs="0"/>
      </sequence>
      <attribute name="Recipient" type="string" use="optional"/>
    </extension>
  </complexContent>
</complexType>
```

`ReferenceList` is an optional element containing pointers to data and keys encrypted using this key. The reference list may contain multiple references to `EncryptedKey` and `EncryptedData` elements. This is done using `KeyReference` and `DataReference` elements respectively. These are defined below.

`CarriedKeyName` is an optional element for associating a user readable name with the key value. This may then be used to reference the key using the `ds:KeyName` element within `ds:KeyInfo`. The same `CarriedKeyName` label, unlike an ID type, may occur multiple times within a single document. The value of the key MUST be the same in all `EncryptedKey` elements identified with the same `CarriedKeyName` label within a single XML document. Note that because whitespace is significant in the value of the `ds:KeyName` element, whitespace is also significant in the value of the `CarriedKeyName` element.

`Recipient` is an optional attribute that contains a hint as to which recipient this encrypted key value is intended for. Its contents are application dependent.

The `Type` attribute inherited from `EncryptedType` can be used to further specify the type of the encrypted key if the `EncryptionMethod Algorithm` does not define a unambiguous encoding/representation. (Note, all the algorithms in this specification have an unambiguous representation for their associated key structures.)

3.5.2 The `DerivedKey` Element

Identifier

`Type="http://www.w3.org/2009/xmlenc11#DerivedKey"`

(This can be used within a `ds:RetrievalMethod` element to identify the referent's type.)

The `DerivedKey` element is used to transport information about a derived key from the originator to recipient(s). It may be used as a stand-alone XML document, be placed within an application document, or appear inside an `EncryptedData` or `Signature` element as a child of a `ds:KeyInfo` element. The key value itself is never sent by the originator. Rather, the originator provides information to the recipient(s) by which the recipient(s) can derive the same key value. When the key has been derived the resulting octets are made available to the `EncryptionMethod` or `SignatureMethod` algorithm without any additional processing.

```
Schema Definition:

<!-- targetNamespace=&#x27;http://www.w3.org/2009/xmlenc11#&#x27; -->

<element name="DerivedKey" type="xenc11:DerivedKeyType"/>

<complexType name="DerivedKeyType">
  <sequence>
    <element ref="xenc11:KeyDerivationMethod" minOccurs="0"/>
    <element ref="xenc:ReferenceList" minOccurs="0"/>
    <element name="DerivedKeyName" type="string" minOccurs="0"/>
    <element name="MasterKeyName" type="string" minOccurs="0"/>
  </sequence>
  <attribute name="Recipient" type="string" use="optional"/>
  <attribute name="Id" type="ID" use="optional"/>
  <attribute name="Type" type="anyURI" use="optional"/>
</complexType>

<element name="KeyDerivationMethod" type="xenc:KeyDerivationMethodType"/>

<complexType name="KeyDerivationMethodType">
  <sequence>
    <any namespace="##any" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Algorithm" type="anyURI" use="required"/>
</complexType>
```

`KeyDerivationMethod` is an optional element that describes the key derivation algorithm applied to the master (underlying) key material. If the element is absent, the key derivation algorithm must be known by the recipient or the recipient's key derivation will fail.

ReferenceList is an optional element containing pointers to data and keys encrypted using this key. The reference list may contain multiple references to EncryptedKey or EncryptedData elements. This is done using KeyReference and DataReference elements from XML Encryption.

The optional DerivedKeyName element is used to identify the derived key value. This element may then be referenced by the ds:KeyName element in ds:KeyInfo. The same DerivedKeyName label, unlike an ID type, may occur multiple times within a single document. Note that because whitespace is significant in the value of the ds:KeyName element, whitespace is also significant in the value of the DerivedKeyName element.

MasterKeyName is an optional element for associating a user readable name with the master key (or secret) value. The same MasterKeyName label, unlike an ID type, may occur multiple times within a single document. The value of the master key MUST be the same in all DerivedKey elements identified with the same MasterKeyName label within a single XML document. If no MasterKeyName is provided, the master key material must be known by the recipient or key derivation will fail.

Recipient is an optional attribute that contains a hint as to which recipient this derived key value is intended for. Its contents are application dependent.

The optional Id attribute provides for the standard method of assigning a string id to the element within the document context.

The Type attribute can be used to further specify the type of the derived key if the KeyDerivationMethod algorithm does not define an unambiguous encoding/representation.

3.5.3 The ds:RetrievalMethod Element

The ds:RetrievalMethod [XMLDSIG-CORE1] with a Type of 'http://www.w3.org/2001/04/xmlenc#EncryptedKey' provides a way to express a link to an EncryptedKey element containing the key needed to decrypt the CipherData associated with an EncryptedData or EncryptedKey element. The ds:RetrievalMethod [XMLDSIG-CORE1] with a Type of 'http://www.w3.org/2001/04/xmlenc#DerivedKey' provides a way to express a link to a DerivedKey element used to derive the key needed to decrypt the CipherData associated with an EncryptedData or EncryptedKey element. The ds:RetrievalMethod with one of these types is always a child of the ds:KeyInfo element and may appear multiple times. If there is more than one instance of a ds:RetrievalMethod in a ds:KeyInfo of this type, then the EncryptedKey objects referred to must contain the same key value, possibly encrypted in different ways or for different recipients.

3.6 The ReferenceList Element

ReferenceList is an element that contains pointers from a key value of an EncryptedKey or DerivedKey to items encrypted by that key value (EncryptedData or EncryptedKey elements).

```
Schema Definition:

<element name="ReferenceList">
  <complexType>
    <choice minOccurs="1" maxOccurs="unbounded">
      <element name="DataReference" type="xenc:ReferenceType"/>
      <element name="KeyReference" type="xenc:ReferenceType"/>
    </choice>
  </complexType>
</element>

<complexType name="ReferenceType">
  <sequence>
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="URI" type="anyURI" use="required"/>
</complexType>
```

DataReference elements are used to refer to EncryptedData elements that were encrypted using the key defined in the enclosing EncryptedKey or DerivedKey element. Multiple DataReference elements can occur if multiple EncryptedData elements exist that are encrypted by the same key.

KeyReference elements are used to refer to EncryptedKey elements that were encrypted using the key defined in the enclosing EncryptedKey or DerivedKey element. Multiple KeyReference elements can occur if multiple EncryptedKey elements exist that are encrypted by the same key.

For both types of references one may optionally specify child elements to aid the recipient in retrieving the EncryptedKey and/or EncryptedData elements. These could include information such as XPath transforms, decompression transforms, or information on how to retrieve the elements from a document storage facility. For example:

EXAMPLE 14

```
<ReferenceList>
  <DataReference URI="#invoice34">
    <ds:Transforms>
      <ds:Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116">
        <ds:XPath xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
          self::xenc:EncryptedData[@Id="example1"]
        </ds:XPath>
      </ds:Transform>
    </ds:Transforms>
  </DataReference>
</ReferenceList>
```

3.7 The EncryptionProperties Element

Identifier

Type="http://www.w3.org/2001/04/xmlenc#EncryptionProperties"

(This can be used within a ds:Reference element to identify the referent's type.)

Additional information items concerning the generation of the EncryptedData or EncryptedKey can be placed in an EncryptionProperty element (e.g., date/time stamp or the serial number of cryptographic hardware used during encryption). The Target attribute identifies the EncryptedType structure being described. anyAttribute permits the inclusion of attributes from the XML namespace to be included (i.e., xml:space, xml:lang, and xml:base).

```
Schema Definition:

<element name="EncryptionProperties" type="xenc:EncryptionPropertiesType"/>

<complexType name="EncryptionPropertiesType">
  <sequence>
    <element ref="xenc:EncryptionProperty" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>

<element name="EncryptionProperty" type="xenc:EncryptionPropertyType"/>

<complexType name="EncryptionPropertyType" mixed="true">
  <choice maxOccurs="unbounded">
    <any namespace="##other" processContents="lax"/>
  </choice>
  <attribute name="Target" type="anyURI" use="optional"/>
  <attribute name="Id" type="ID" use="optional"/>
  <anyAttribute namespace="http://www.w3.org/XML/1998/namespace"/>
</complexType>
```

4. Processing Rules

This section describes the operations to be performed as part of encryption and decryption processing by implementations of this specification. The conformance requirements are specified over the following roles:

Encryptor
: An XML Encryption implementation with the role of encrypting data.

Decryptor
: An XML Encryption implementation with the role of decrypting data.

Encryptor and Decryptor are invoked by the Application. This specification does not include normative definitions for application behavior. However, this specification does include conformance requirements on encrypted data that may only be achievable through appropriate behavior by all three parties. It is up to specific deployment contexts how this is achieved.

4.1 Intended Application Model

The processing rules for XML Encryption are designed around an intended application model that this version of the specification does not cover normatively.

In the intended processing model, XML Encryption is used to encrypt an octet-stream, an EXI stream, or a fragment of an XML document that matches either the content or element production from [XML10].

If XML Encryption is used with some octet-stream, the precise encoding and meaning of that octet-stream is up to the application, but treated as opaque by the Encryptor or Decryptor. The application may use the Type, Encoding and MimeType parameters to transport further information about the nature of that octet-stream. Hence, an unknown Type parameter is, in general, not treated as an error by either the Encryptor or Decryptor, but instead simply passed through, along with the other relevant parameters and the cleartext octet-stream.

Si se utiliza el cifrado XML con XML elemento XML content, los cifradores y descifradores suelen realizar un procesamiento específico del tipo:

- Si un elementestá cifrado, entonces Encryptor reemplazará el elemento en cuestión con un EncryptedDataelemento construido apropiadamente. El Decryptor, por el contrario, reemplazará el EncryptedDataelemento con su texto sin cifrar.
- Si XML contentestá cifrado, el Encryptor también reemplazará este contenido con un EncryptedDataelemento construido apropiadamente y el Decryptor revertirá esta operación.

Tenga en cuenta que el comportamiento previsto de Encryptor a menudo hará que el documento con partes cifradas deje de ser válido con respecto a su esquema para el formato XML de alojamiento, a menos que ese formato esté específicamente preparado para usarse con XML Encryption. NO SE REQUIERE un cifrador o descifrador que implemente el modelo de procesamiento previsto para garantizar que el XML resultante tenga un esquema válido para el formato XML de alojamiento.

Si el procesamiento XML se maneja dentro de Encryptor y Decryptor, y se utilizan los Typevalores de atributos para elementy texto sin cifrar, entonces Encryptor y Decryptor DEBEN garantizar que el texto sin cifrar XML se serialice como UTF-8 antes del cifrado y, si es necesario, se vuelva a convertir. a cualquier otra codificación que pueda ser utilizada por el contexto XML circundante.content

Si se utiliza el cifrado XML con una secuencia EXI [EXI], los cifradores y descifradores procesan el contenido como para el procesamiento de elementos XML o contenido XML, pero teniendo en cuenta la serialización EXI. En particular, el cifrador reemplazará el elemento XML o el fragmento XML en cuestión con un elemento EncryptedData construido adecuadamente. Por el contrario, Decryptor reemplazará el elemento EncryptedData con su elemento XML de texto sin formato o fragmento XML. Tenga en cuenta que el documento XML en el que está incrustado el elemento EncryptedData puede codificarse usando EXI y/o EXI puede usarse para codificar el texto sin cifrar antes del cifrado.

4.2Type Valores de parámetros conocidos

PARA FINES DE INTEROPERABILIDAD, DEBEN implementarse los siguientes tipos de modo que una implementación pueda tomar como entrada y producir como salida datos que coincidan con las reglas de producción 39 y 43 de [XML10]:

elemento ' **http://www.w3.org/2001/04/xmlenc#Element** '
: "[39] elemento ::= EmptyElemTag | Contenido de STag ETag "

contenido ' **http://www.w3.org/2001/04/xmlenc#Content** '
: "[43] contenido ::= CharData? ((elemento | Referencia | CDSect | PI | Comentario) CharData?)*"

La compatibilidad con el siguiente tipo es OPCIONAL para Encryptors y Decryptors:

http://www.w3.org/2009/xmlenc11#EXI
: La presencia de esto `Type`indica que el texto sin cifrar es una secuencia EXI [EXI]. Los cifradores y descifradores que admiten este tipo PUEDEN operar directamente en (partes de) flujos EXI.

Los cifradores y descifradores DEBEN manejar valores de atributos desconocidos o vacíos `Type`como señal de que el texto sin cifrar debe manejarse como un flujo de octetos opaco, cuyo procesamiento específico depende de la aplicación que lo invoca. En este caso, los parámetros `Type`, `MimeType`y DEBEN tratarse como datos opacos cuyo procesamiento apropiado depende de la aplicación.`Encoding`

4.3 Cifrado

La selección del algoritmo, los parámetros y las claves de cifrado está fuera del alcance de esta especificación.

Se supone que los datos en texto claro están presentes como un tren de octetos. Si el texto sin formato es de tipo `element`o `content`, los datos DEBEN serializarse en UTF-8 como se especifica en [XML10], utilizando la forma normal C [NFC].

Para que cada elemento de datos se cifre como elemento `EncryptedData`o `EncryptedKey`, el **cifrador** DEBE :

1. Obtener (o derivar) y (opcionalmente) representar la clave.
    1. Si la clave debe identificarse (mediante nombre, URI o incluirse en un elemento secundario), construya la clave `ds:KeyInfo`según corresponda (p. ej. `ds:KeyName`, `ds:KeyValue`, `ds:RetrievalMethod`, etc.)
    2. Si se va a cifrar la clave en sí, construya un `EncryptedKey`elemento aplicando recursivamente este proceso de cifrado. El resultado puede ser hijo de `ds:KeyInfo`o puede existir en otro lugar y puede identificarse en el paso anterior.
    3. Si la clave se derivó de una clave maestra, construya un `DerivedKey`elemento con elementos secundarios asociados. El resultado puede, como en este `EncryptedKey` caso, ser hijo de `ds:KeyInfo`o puede existir en otro lugar.
2. Cifre los datos:
    1. Cifre los octetos utilizando el algoritmo y la clave.
    2. Unless the **decryptor** will implicitly know the type of the encrypted data, the **encryptor** SHOULD set the `Type` to indicate the intended interpretation of the cleartext data. See section 4.2 Well-known Type parameter values for known parameter values.

       If the data is a simple octet sequence it MAY be described with the `MimeType` and `Encoding` attributes. For example, the data might be an XML document (`MimeType="text/xml"`), sequence of characters (`MimeType="text/plain"`), or binary image data (`MimeType="image/png"`).
3. Build the `EncryptedData` or `EncryptedKey` structure:

   An `EncryptedData` or `EncryptedKey` structure represents all of the information previously discussed including the type of the encrypted data, encryption algorithm, parameters, key, type of the encrypted data, etc.
    1. If the encrypted octet sequence obtained in step 2 is to be stored in the `CipherData` element within the `EncryptedData` or `EncryptedKey` element, then the base64 representation of the encrypted octet sequence is inserted as the content of a `CipherValue` element.
    2. If the encrypted octet sequence is stored externally to the `EncryptedData` or `EncryptedKey` element, then the URI and transforms (if any) required for the Decryptor to retrieve the encrypted octet sequence are described within a `CipherReference` element.

4.4 Decryption

For each `EncryptedData` or `EncryptedKey` to be decrypted, the **decryptor** MUST:

1. Determine the algorithm, parameters and key information to be used. This information may be obtained out-of-band, or determined according to a `ds:KeyInfo` element; see section 3.5 Extensions to ds:KeyInfo Element.
2. Decrypt the data contained in the `CipherData` element.
    1. If a `CipherValue` child element is present, then the associated text value is retrieved and base64 decoded so as to obtain the encrypted octet sequence.
    2. If a `CipherReference` child element is present, the URI and transforms (if any) are used to retrieve the encrypted octet sequence.
    3. The encrypted octet sequence is decrypted using the algorithm, parameters and key value already determined from step 1.

4.5 XML Encryption

Encryption and decryption operations are operations on octets. The **application** is responsible for the marshalling XML such that it can be serialized into an octet sequence, encrypted, decrypted, and be of use to the recipient.

For example, if the application wishes to canonicalize its data or encode/compress the data in an XML packaging format, the application needs to marshal the XML accordingly and identify the resulting type via the `EncryptedData` `Type` attribute. The likelihood of successful decryption and subsequent processing will be dependent on the recipient's support for the given type. Also, if the data is intended to be processed both before encryption and after decryption (e.g., XML Signature [XMLDSIG-CORE1] validation or an XSLT transform) the encrypting application must be careful to preserve information necessary for that process's success.

The following sections contain specifications for decrypting, replacing, and serializing XML content (i.e., `Type` 'element' or element 'content') using the [XPATH] data model. These sections are non-normative and OPTIONAL to implementers of this specification, but they may be normatively referenced by and be required by other specifications that require a consistent processing for applications, such as [XMLENC-DECRYPT].

4.5.1 A Decrypt Implementation (Non-normative)

Where *P* is the context in which the serialized XML should be parsed (a document node or element node) and *O* is the octet sequence representing UTF-8 encoded characters resulting from step 4.3 in section 4.4 Decryption. *Y* is node-set representing the decrypted content obtained by the following steps:

1. Let *C* be the parsing context of a child of *P*, which consists of the following items:
    - Prefix and namespace name of each namespace that is in scope for *P*.
    - Name and value of each general entity that is effective for the XML document causing *P*.
2. Wrap the decrypted octet stream *O* in the context *C* as specified in section 4.5.4 Text Wrapping.
3. Parse the wrapped octet stream as described in The Reference Processing Model (section 4.3.3.2) of [XMLDSIG-CORE1], resulting in a node-set.
4. *Y* is the node-set obtained by removing the root node, the wrapping element node, and its associated set of attribute and namespace nodes from the node-set obtained in Step 3.

4.5.2 A Decrypt and Replace Implementation (Non-normative)

Where *X* is the [XPATH] node set corresponding to an XML document and *e* is an `EncryptedData` element node in *X*.

1. *Z* is an [XPATH] node-set that identical to X except where the element node *e* is an `EncryptedData` element type. In which case:
    1. Decrypt *e* in the context of its parent node as specified in the section 4.5.1 A Decrypt Implementation (Non-normative) yielding *Y*, an [XPATH] node set.
    2. Include *Y* in place of *e* and its descendants in *X*. Since [XPATH] does not define methods of replacing node-sets from different documents, the result MUST be equivalent to replacing e with the octet stream resulting from its decryption in the serialized form of *X* and re-parsing the document. However, the actual method of performing this operation is left to the implementor.

4.5.3 Serializing XML (Non-normative)

4.5.3.1 Default Namespace Considerations

In section 4.3 Encryption (step 3.1), when serializing an XML fragment special care SHOULD be taken with respect to default namespaces. If the data will be subsequently decrypted in the context of a parent XML document then serialization can produce elements in the wrong namespace. Consider the following fragment of XML:

EXAMPLE 15

```
<Document xmlns="http://example.org/">
  <ToBeEncrypted xmlns="" />
</Document>
```

Serialization of the element `ToBeEncrypted` fragment via [XML-C14N] would result in the characters "`<ToBeEncrypted></ToBeEncrypted>`" as an octet stream. The resulting encrypted document would be:

EXAMPLE 16

```
<Document xmlns="http://example.org/">
  <EncryptedData xmlns="...">
    <!-- Containing the encrypted "<ToBeEncrypted></ToBeEncrypted>" -->
  </EncryptedData>
</Document>
```

Decrypting and replacing the `EncryptedData` within this document would produce the following incorrect result:

EXAMPLE 17

```
<Document xmlns="http://example.org/">
  <ToBeEncrypted/>
</Document>
```

This problem arises because most XML serializations assume that the serialized data will be parsed directly in a context where there is no default namespace declaration. Consequently, they do not redundantly declare the empty default namespace with an `xmlns=""`. If, however, the serialized data is parsed in a context where a default namespace declaration is in scope (e.g., the parsing context as described in section 4.5.1 A Decrypt Implementation (Non-normative)), then it may affect the interpretation of the serialized data.

To solve this problem, a canonicalization algorithm MAY be augmented as follows for use as an XML encryption serializer:

- A default namespace declaration with an empty value (i.e., `xmlns=""`) SHOULD be emitted where it would normally be suppressed by the canonicalization algorithm.

While the result may not be in proper canonical form, this is harmless as the resulting octet stream will not be used directly in a [XMLDSIG-CORE1] signature value computation. Returning to the preceding example with our new augmentation, the `ToBeEncrypted` element would be serialized as follows:

```
<ToBeEncrypted xmlns=""></ToBeEncrypted>
```

When processed in the context of the parent document, this serialized fragment will be parsed and interpreted correctly.

This augmentation can be retroactively applied to an existing canonicalization implementation by canonicalizing each apex node and its descendants from the node set, inserting `xmlns=""` at the appropriate points, and concatenating the resulting octet streams.

4.5.3.2 XML Attribute Considerations

Similar attention between the relationship of a fragment and the context into which it is being inserted should be given to the xml:base, xml:lang, and xml:space attributes as mentioned in the Security Considerations of [XML-EXC-C14N]. For example, if the element:

EXAMPLE 18

```
<Bongo href="example.xml"/>
```

is taken from a context and serialized with no xml:base [XMLBASE] attribute and parsed in the context of the element:

EXAMPLE 19

```
<Baz xml:base="http://example.org/"/>
```

the result will be:

EXAMPLE 20

```
<Baz xml:base="http://example.org/"><Bongo href="example.xml"/></Baz>
```

Bongo's href is subsequently interpreted as "http://example.org/example.xml". If this is not the correct URI, Bongo should have been serialized with its own xml:base attribute.

Unfortunately, the recommendation that an empty value be emitted to divorce the default namespace of the fragment from the context into which it is being inserted cannot be made for the attributes xml:base, and xml:space. (Error 41 of the XML 1.0 Second Edition Specification Errata clarifies that an empty string value of the attribute xml:lang *is* considered as if, "there is no language information available, just as if xml:lang had not been specified".)The interpretation of an empty value for the xml:base or xml:space attributes is undefined or maintains the contextual value. Consequently, applications SHOULD ensure (1) fragments that are to be encrypted are not dependent on XML attributes, or (2) if they are dependent and the resulting document is intended to be valid [XML10], the fragment's definition permits the presence of the attributes and that the attributes have non-empty values.

4.5.4 Text Wrapping

This section specifies the process for wrapping text in a given parsing context. The process is based on the proposal by Richard Tobin [Tobin] for constructing the infoset [XML-INFOSET] of an external entity.

The process consists of the following steps:

1. If the parsing context contains any general entities, then emit a document type declaration that provides entity declarations.
2. Emit a dummy element start-tag with namespace declaration attributes declaring all the namespaces in the parsing context.
3. Emit the text.
4. Emit a dummy element end-tag.

In the above steps, the document type declaration and dummy element tags MUST be encoded in UTF-8.

Consider the following document containing an EncryptedData element:

EXAMPLE 21

```
<!DOCTYPE Document [
<!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">
]>
<Document xmlns="http://example.org/">
  <foo:Body xmlns:foo="http://example.org/foo">
    <EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
                   Type="http://www.w3.org/2001/04/xmlenc#Element">
      ...
    </EncryptedData>
  </foo:Body>
</Document>
```

If the EncryptedData element is decrypted to the text "<One><foo:Two/></One>", then the wrapped form is as follows:

EXAMPLE 22

```
<!DOCTYPE dummy [
<!ENTITY dsig "http://www.w3.org/2000/09/xmldsig#">
]>
<dummy xmlns="http://example.org/"
       xmlns:foo="http://example.org/foo">
  <One>
    <foo:Two/>
  </One>
</dummy>
```

5. Algorithms

This section discusses algorithms used with the XML Encryption specification. Entries contain the identifier to be used as the value of the Algorithm attribute of the EncryptionMethod element or other element representing the role of the algorithm, a reference to the formal specification, definitions for the representation of keys and the results of cryptographic operations where applicable, and general applicability comments.

5.1 Algorithm Identifiers and Implementation Requirements

All algorithms listed below have implicit parameters depending on their role. For example, the data to be encrypted or decrypted, keying material, and direction of operation (encrypting or decrypting) for encryption algorithms. Any explicit additional parameters to an algorithm appear as content elements within the element. Such parameter child elements have descriptive element names, which are frequently algorithm specific, and SHOULD be in the same namespace as this XML Encryption specification, the XML Signature specification, or in an algorithm specific namespace. An example of such an explicit parameter could be a nonce (unique quantity) provided to a key agreement algorithm.

This specification defines a set of algorithms, their URIs, and requirements for implementation. Levels of requirement specified, such as "REQUIRED" or "OPTIONAL", refer to implementation, not use. Furthermore, the mechanism is extensible, and alternative algorithms may be used.

5.1.1 Table of Algorithms

La siguiente tabla enumera las categorías de algoritmos. Dentro de cada categoría, se proporciona un nombre breve, el nivel de requisito de implementación y un URI de identificación para cada algoritmo.

Cifrado de bloques

1. TRIPLEDES REQUERIDOS
   http://www.w3.org/2001/04/xmlenc#tripledes-cbc
2. AES-128 REQUERIDO
   http://www.w3.org/2001/04/xmlenc#aes128-cbc
3. AES-256 REQUERIDO
   http://www.w3.org/2001/04/xmlenc#aes256-cbc
4. AES128-GCM REQUERIDO
   http://www.w3.org/2009/xmlenc11#aes128-gcm
5. OPCIONAL AES-192
   http://www.w3.org/2001/04/xmlenc#aes192-cbc
6. OPCIONAL AES192-GCM
   http://www.w3.org/2009/xmlenc11#aes192-gcm
7. OPCIONAL AES256-GCM
   http://www.w3.org/2009/xmlenc11#aes256-gcm

Nota: Se recomienda encarecidamente el uso de AES GCM sobre cualquier algoritmo de cifrado de bloques CBC, ya que los avances recientes en criptoanálisis [XMLENC-CBC-ATTACK][XMLENC-CBC-ATTACK-COUNTERMEASURES] han puesto en duda la capacidad de los algoritmos de cifrado de bloques CBC para proteger datos simples. texto cuando se utiliza con cifrado XML. Se deben considerar otras mitigaciones al utilizar el cifrado de bloques CBC, como transmitir los datos cifrados a través de un canal seguro como TLS. Los algoritmos de cifrado de bloques CBC que se enumeran como necesarios siguen siéndolo por motivos de compatibilidad con versiones anteriores.

Cifrado de flujo

1. none
   A continuación se proporcionan sintaxis y recomendaciones para admitir algoritmos especificados por el usuario.

Derivación clave

1. REQUERIDO ConcatKDF
   http://www.w3.org/2009/xmlenc11#ConcatKDF
2. PBKDF2 OPCIONAL
   http://www.w3.org/2009/xmlenc11#pbkdf2

Transporte clave

1. RSA-OAEP REQUERIDO (INCLUIDO MGF1 CON SHA1)
   http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p
2. RSA-OAEP opcional
   http://www.w3.org/2009/xmlenc11#rsa-oaep
3. OPCIONAL RSA-v1.5 (consulte la nota de seguridad RSA-v1.5)
   http://www.w3.org/2001/04/xmlenc#rsa-1_5

Acuerdo clave

1. OBLIGATORIO Curva Elíptica Diffie-Hellman (Modo Efímero-Estático)
   http://www.w3.org/2009/xmlenc11#ECDH-ES
2. Acuerdo de clave Diffie-Hellman OPCIONAL (MODO EFÍMERO-ESTÁTICO) CON FUNCIÓN DE DERIVACIÓN DE CLAVE HEREDADA
   http://www.w3.org/2001/04/xmlenc#dh
3. ACUERDO DE CLAVES DIFFIE-HELLMAN OPCIONAL (modo Efímero-Estático) con funciones explícitas de derivación de claves
   http://www.w3.org/2009/xmlenc11#dh-es

Envoltura de clave simétrica

1. TRIPLEDES REQUERIDOS KEYWRAP
   http://www.w3.org/2001/04/xmlenc#kw-tripledes
2. REQUERIDO AES-128 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes128
3. REQUERIDO AES-256 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes256
4. OPCIONAL AES-192 KeyWrap
   http://www.w3.org/2001/04/xmlenc#kw-aes192

Resumen del mensaje

1. REQUERIDO SHA1 (*Se DESACONSEJA su uso* ; ver más abajo).
   http://www.w3.org/2000/09/xmldsig#sha1
2. SHA256 REQUERIDO
   http://www.w3.org/2001/04/xmlenc#sha256
3. SHA384 OPCIONAL
   http://www.w3.org/2001/04/xmlenc#sha384
4. SHA512 OPCIONAL
   http://www.w3.org/2001/04/xmlenc#sha512
5. RIPEMD-160 OPCIONAL
   http://www.w3.org/2001/04/xmlenc#ripemd160

Canonicalización

1. OPCIONAL XML canónico 1.0 (omitir comentarios)
   http://www.w3.org/TR/2001/REC-xml-c14n-20010315
2. OPCIONAL Canonical XML 1.0 (con comentarios)
   http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
3. OPCIONAL XML canónico 1.1 (omitir comentarios)
   http://www.w3.org/2006/12/xml-c14n11
4. OPCIONAL XML canónico 1.1 (con comentarios)
   http://www.w3.org/2006/12/xml-c14n11#WithComments
5. OPCIONAL Canonicalización XML exclusiva 1.0 (omitir comentarios)
   http://www.w3.org/2001/10/xml-exc-c14n#
6. OPCIONAL Canonicalización XML exclusiva 1.0 (con comentarios)
   http://www.w3.org/2001/10/xml-exc-c14n#WithComments

Codificación

1. REQUIERE base64 (*nota)
   http://www.w3.org/2000/09/xmldsig#base64

Transforma

1. REQUIERE base64 (*nota)
   http://www.w3.org/2000/09/xmldsig#base64

*nota: El mismo URI se utiliza para identificar base64 tanto en el contexto de "codificación" (por ejemplo, cuando se utiliza con el `Encoding`atributo de un `EncryptedKey`elemento, consulte la sección 3.1 El elemento EncryptedType) como en el contexto de "transformación" (cuando se identifica una transformación base64 para a `CipherReference`, consulte la sección 3.3.1 El elemento CipherReference).

5.2 Algoritmos de cifrado de bloques

Los algoritmos de cifrado de bloques están diseñados para cifrar y descifrar datos en bloques de varios octetos de tamaño fijo. Sus identificadores aparecen como el valor de los `Algorithm`atributos de `EncryptionMethod`los elementos hijos de `EncryptedData`.

Nota : Los algoritmos de cifrado de bloques CBC no deben utilizarse sin tener en cuenta posibles riesgos de seguridad graves .

Block encryption algorithms take, as implicit arguments, the data to be encrypted or decrypted, the keying material, and their direction of operation. For all of these algorithms specified below, an initialization vector (IV) is required that is encoded with the cipher text. For user specified block encryption algorithms, the IV, if any, could be specified as being with the cipher data, as an algorithm content element, or elsewhere.

The IV is encoded with and before the cipher text for the algorithms below for ease of availability to the decryption code and to emphasize its association with the cipher text. Good cryptographic practice requires that a different IV be used for every encryption.

5.2.1 Padding

Since the data being encrypted is an arbitrary number of octets, it may not be a multiple of the block size. This is solved by padding the plain text up to the block size before encryption and unpadding after decryption. The padding algorithm is to calculate the smallest non-zero number of octets, say `N`, that must be suffixed to the plain text to bring it up to a multiple of the block size. We will assume the block size is `B` octets so `N` is in the range of 1 to `B`. Pad by suffixing the plain text with `N-1` arbitrary pad bytes and a final byte whose value is `N`. On decryption, just take the last byte and, after sanity checking it, strip that many bytes from the end of the decrypted cipher text.

For example, assume an 8 byte block size and plain text of `0x616263`. The padded plain text would then be `0x616263????????05` where the "??" bytes can be any value. Similarly, plain text of `0x2122232425262728` would be padded to `0x2122232425262728??????????????08`.

5.2.2 Triple DES

Identifier:
 http://www.w3.org/2001/04/xmlenc#tripledes-cbc

NIST SP800-67 [SP800-67] specifies three sequential FIPS 46-3 [DES] operations. The XML Encryption TRIPLEDES consists of a DES encrypt, a DES decrypt, and a DES encrypt used in the Cipher Block Chaining (CBC) mode with 192 bits of key and a 64 bit Initialization Vector (IV). Of the key bits, the first 64 are used in the first DES operation, the second 64 bits in the middle DES operation, and the third 64 bits in the last DES operation.

Nota: Cada uno de estos 64 bits de clave contiene 56 bits efectivos y 8 bits de paridad. Por tanto, sólo hay 168 bits operativos de los 192 que se transportan para una clave TRIPLEDES. (Dependiendo del criterio utilizado para el análisis, se puede pensar que la fuerza efectiva de la clave es de 112 bits (debido a los ataques intermedios) o incluso menos).

El texto cifrado resultante tiene el prefijo IV. Si se incluye en la salida XML, está codificado en base64. Un ejemplo de método de cifrado TRIPLEDES es el siguiente:

EJEMPLO 23

```
< Algoritmo del método de cifrado = "http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
```

Nota : Los algoritmos de cifrado de bloques CBC no deben utilizarse sin tener en cuenta posibles riesgos de seguridad graves .

5.2.3 AES

Identificador:
http://www.w3.org/2001/04/xmlenc#aes128-cbc
http://www.w3.org/2001/04/xmlenc#aes192-cbc
http://www.w3.org/2001/04/xmlenc#aes256-cbc

[AES] se utiliza en el modo Cipher Block Chaining (CBC) con un vector de inicialización (IV) de 128 bits. El texto cifrado resultante tiene el prefijo IV. Si se incluye en la salida XML, está codificado en base64. Un ejemplo de método de cifrado AES es el siguiente:

EJEMPLO 24

```
< Algoritmo del método de cifrado = "http://www.w3.org/2001/04/xmlenc#aes128-cbc" />
```

Nota : Los algoritmos de cifrado de bloques CBC no deben utilizarse sin tener en cuenta posibles riesgos de seguridad graves .

5.2.4 AES-GCM

Identificador:
http://www.w3.org/2009/xmlenc11#aes128-gcm
http://www.w3.org/2009/xmlenc11#aes192-gcm
http://www.w3.org/2009/xmlenc11#aes256-gcm

AES-GCM [SP800-38D] es un mecanismo de cifrado autenticado. Equivale a realizar estas dos operaciones en un solo paso: cifrado AES seguido de firma HMAC.

AES-GCM es muy atractivo desde el punto de vista del rendimiento porque el costo de AES-GCM es similar al del cifrado AES-CBC normal, pero logra el mismo resultado que el cifrado y la firma HMAC. También se puede canalizar AES-GCM para que sea susceptible de aceleración por hardware.

A los efectos de esta especificación, AES-GCM se utilizará con un vector de inicialización (IV) de 96 bits y una etiqueta de autenticación (T) de 128 bits. El texto cifrado contiene primero el IV, seguido de los octetos cifrados y finalmente la etiqueta de autenticación. No se debe utilizar ningún relleno durante el cifrado. Durante el descifrado, la implementación debe comparar la etiqueta de autenticación calculada durante el descifrado con la etiqueta de autenticación especificada y fallar si no coinciden. Para obtener detalles sobre la implementación de AES-GCM, consulte [SP800-38D].

5.3 Algoritmos de cifrado de flujo

Los algoritmos de cifrado de flujo simple generan, en función de la clave, un flujo de bytes a los que se aplica XOR con los bytes de datos de texto sin formato para producir el texto cifrado en el cifrado y con los bytes de texto cifrado para producir texto sin formato al descifrar. Normalmente se utilizan para el cifrado de datos y se especifican por el valor del `Algorithm`atributo del `EncryptionMethod`hijo de un `EncryptedData` elemento.

NOTA: Es fundamental que cada clave de cifrado de flujo simple (o clave y vector de inicialización (IV) si también se usa un IV) se use solo una vez. Si alguna vez se usa la misma clave (o clave y IV) en dos mensajes, al aplicar XOR en los dos textos cifrados, puede obtener el XOR de los dos textos sin formato. Esto suele ser muy comprometedor.

En este documento no se especifican algoritmos de cifrado de flujo específicos, pero esta sección se incluye para proporcionar pautas generales.

Los algoritmos de transmisión suelen utilizar el `KeySize`parámetro explícito opcional. En los casos en los que el tamaño de la clave no sea evidente en el URI del algoritmo o en el origen de la clave, como en el uso de métodos de acuerdo de claves, este parámetro establece el tamaño de la clave. Si el tamaño de la clave que se utilizará es evidente y no está de acuerdo con el parámetro, DEBE devolverse `KeySize`un error . La implementación de cualquier algoritmo de flujo es opcional. El esquema para el parámetro KeySize es el siguiente:

```
Definición del esquema :
<simpleType nombre = "KeySizeType" > <restricción base = "entero" /> </simpleType>
```

5.4 Derivación de claves

La derivación de claves es un mecanismo bien establecido para generar nuevo material de claves criptográficas a partir de algún material de claves original ("maestro") existente y potencialmente de otra información. Las claves derivadas se utilizan para diversos fines, incluido el cifrado de datos y la autenticación de mensajes. La razón para realizar la derivación de claves en sí suele ser una combinación del deseo de ampliar un conjunto determinado, pero limitado, de material de claves original y prácticas de seguridad prudentes para limitar el uso (exposición) de dicho material de claves. La separación de claves (como evitar el uso del mismo material clave para múltiples propósitos) es un ejemplo de tales prácticas.

El proceso de derivación de claves puede basarse en frases de contraseña acordadas o recordadas por los usuarios, o puede basarse en algunas claves criptográficas "maestras" compartidas (y tener como objetivo reducir la exposición de dichas claves maestras), etc. Se pueden utilizar las propias claves derivadas. en Firma XML y Cifrado XML como cualquier otra clave; en particular, pueden usarse para calcular códigos de autenticación de mensajes (por ejemplo, firmas digitales que utilizan claves simétricas) o para fines de cifrado/descifrado.

5.4.1 ConcatKDF

Identificador:
http://www.w3.org/2009/xmlenc11#ConcatKDF

El algoritmo de derivación de claves ConcatKDF, definido en la Sección 5.8.1 de NIST SP 800-56A [SP800-56A] (y equivalente a la función KDF3 definida en ANSI X9.44-2007 [ANSI-X9-44-2007] cuando el contenido del `OtherInfo`parámetro está estructurado como en NIST SP 800-56A), toma varios parámetros. Estos parámetros están representados en `xenc11:ConcatKDFParamsType`:

```
Definición del esquema :

<!-- targetNamespace='http://www.w3.org/2009/xmlenc11#' -->

<!-- use este tipo de elemento como hijo de xenc11:KeyDerivationMethod
     cuando se usa con ConcatKDF --> < nombre del elemento = "ConcatKDFParams" tipo = "xenc11:ConcatKDFParamsType" />


<complexType nombre = "ConcatKDFParamsType" > <secuencia> <elemento ref = "ds:DigestMethod" /> </secuencia> <atributo nombre =
```

The `ds:DigestMethod` element identifies the digest algorithm used by the KDF. Compliant implementations MUST support SHA-256 and SHA-1 (support for SHA-1 is present only for backwards-compatibility reasons). Support for SHA-384 and SHA-512 is OPTIONAL.

The `AlgorithmID`, `PartyUInfo`, `PartyVInfo`, `SuppPubInfo` and `SuppPrivInfo` attributes are as defined in [SP800-56A]. Their presence is optional but `AlgorithmID`, `PartyVInfo` and `PartyUInfo` MUST be present for applications that need to comply with [SP800-56A]. Note: The `PartyUInfo` component shall include a nonce when ConcatKDF is used in conjunction with a static-static Diffie-Hellman (or static-static ECDH) key agreement scheme; see further [SP800-56A].

In [SP800-56A], `AlgorithmID`, `PartyUInfo`, `PartyVInfo`, `SuppPubInfo` and `SuppPrivInfo` attributes are all defined as arbitrary-length bitstrings, thus they may need to be padded in order to be encoded into hexBinary for XML Encryption. The following padding and encoding method MUST be used when encoding bitstring values for the `AlgorithmID`, `PartyUInfo`, `PartyVInfo`, `SuppPubInfo` and `SuppPrivInfo`:

1. The bitstring is divided into octets using big-endian encoding. If the length of the bitstring is not a multiple of 8 then add padding bits (value 0) as necessary to the last octet to make it a multiple of 8.

2. Prepend one octet to the octets string from step 1. This octet shall identify (in a big-endian representation) the number of padding bits added to the last octet in step 1.

3. Encode the octet string resulting from step 2 as a hexBinary string.

Example: the bitstring `11011`, which is 5 bits long, gets 3 additional padding bits to become the bitstring `11011000` (or `D8` in hex). This bitstring is then prepended with one octet identifying the number of padding bits to become the octet string (in hex) `03D8`, which then finally is encoded as a hexBinary string value of "03D8".

Note that as specified in [SP800-56A], these attributes shall be concatenated to form a bit string “OtherInfo” that is used with the key derivation function. The concatenation SHALL be done using the original, unpadded bit string values.” Applications MUST also verify that these attributes, in an application-specific way not defined in this document, identify algorithms and parties in accordance with NIST SP800-56.

An example of an `xenc11:DerivedKey` element with this key derivation algorithm given below. In this example, the bitstring value of `AlgorithmID` is `00000000`, the bitstring value of `PartyUInfo` is `11011` and the bitstring value of `PartyVInfo` is `11010`:

EXAMPLE 25

```
<xenc11:DerivedKey
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:xenc11="http://www.w3.org/2009/xmlenc11#">
  <xenc11:KeyDerivationMethod Algorithm="http://www.w3.org/2009/xmlenc11#ConcatKDF">
    <xenc11:ConcatKDFParams AlgorithmID="0000" PartyUInfo="03D8" PartyVInfo="03D0">
      <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    </xenc11:ConcatKDFParams>
  </xenc11:KeyDerivationMethod>
  <xenc:ReferenceList>
    <xenc:DataReference URI="#ED"/>
  </xenc:ReferenceList>
  <xenc11:MasterKeyName>Our other secret</xenc11:MasterKeyName>
</xenc11:DerivedKey>
```

NOTE
While any bit string can be used with ConcatKDF, it is RECOMMENDED to keep byte aligned for greatest interoperability.

5.4.2 PBKDF2

Identifier:
http://www.w3.org/2009/xmlenc11#pbkdf2

El algoritmo de derivación de claves PBKDF2 y las definiciones de tipo ASN.1 para sus parámetros se definen en PKCS #5 v2.0 [PKCS5]. Las definiciones del esquema XML para los parámetros se definen en [PKCS5Amd1] y las mismas se pueden especificar encerrándolas dentro de un `xenc11:PBKDF2-params`elemento secundario del `xenc11:KeyDerivationMethod`elemento.

```
Definición del esquema :
```

```
< nombre del elemento = "PBKDF2-params" tipo = "xenc11:PBKDF2ParameterType" />

<complexType nombre = "PBKDF2ParameterType" > <secuencia> <elemento nombre = "Salt" > <complexType> <elección> <elemento nombr

<complexType nombre = "AlgorithmIdentifierType" > <secuencia> <elemento nombre = "Parámetros" tipo = "anyType" minOccurs = "0"

<complexType nombre = "PRFAlgorithmIdentifierType" > <complexContent> <restricción base = "xenc11:AlgorithmIdentifierType" > <
```

(Nota: se agregó una nueva línea al atributo Algoritmo para que quepa en esta página, pero no forma parte del URI).

El `PBKDF2-params`elemento y sus elementos secundarios tienen los mismos nombres y significados que los componentes correspondientes del `PBKDF2-params`tipo ASN.1 en [PKCS5]. Tenga en cuenta que, en el caso de ConcatKDF y el KDF heredado de Diffie Hellman, `KeyLength`es un parámetro implícito y debe inferirse del contexto, pero en el caso de PBKDF2, el `KeyLength`elemento secundario debe especificarse, ya que se ha convertido en un parámetro obligatorio. consistente con PKCS5. Para PBKDF2, la longitud de la clave inferida debe coincidir con la longitud de la clave especificada; de lo contrario, se trata de una condición de error.

Corresponde `AlgorithmIdentifierType`al `AlgorithmIdentifier`tipo de [PKCS5] y lleva el identificador del algoritmo en el `Algorithm`atributo. Los parámetros específicos del algoritmo, cuando corresponda, se pueden especificar utilizando el `Parameters`elemento.

Se `PRFAlgorithmIdentifierType`deriva de `AlgorithmIdentifierType`y limita la elección de algoritmos a los contenidos en el conjunto PBKDF2-PRF definido en [PKCS5]. Este tipo se utiliza para especificar una función pseudoaleatoria (PRF) para PBKDF2. Mientras que HMAC-SHA1 es el algoritmo PRF predeterminado en [PKCS5], esta especificación RECOMIENDA EL USO DE HMAC-SHA256 (CONSULTE [XMLDSIG-CORE1], [HMAC]).

Un ejemplo de un `xenc11:DerivedKey`elemento con este algoritmo de derivación de claves es:

EJEMPLO 26

```
<xenc11:DerivedKey xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance" xmlns:xenc = "http://www.w3.org/2001/04/xmlenc#"
```

5.5 Transporte clave

Los algoritmos de transporte de claves son algoritmos de cifrado de claves públicas especialmente especificados para cifrar y descifrar claves. Sus identificadores aparecen como `Algorithm`atributos de `EncryptionMethod`elementos hijos de `EncryptedKey`. `EncryptedKey`es a su vez hijo de un `ds:KeyInfo`elemento. El tipo de clave que se transporta, es decir el algoritmo en el que se prevé utilizar la clave transportada, viene dado por el `Algorithm`atributo del `EncryptionMethod`hijo `EncryptedData`o `EncryptedKey`padre de este `ds:KeyInfo`elemento.

(Los algoritmos de transporte de claves se pueden utilizar opcionalmente para cifrar datos, en cuyo caso aparecen directamente como el `Algorithm`atributo de un elemento `EncryptionMethod`secundario `EncryptedData`. Debido a que utilizan algoritmos de clave pública directamente, los algoritmos de transporte de claves no son eficientes para el transporte de ninguna cantidad de datos significativamente. más grandes que las claves simétricas.)

5.5.1 RSA Versión 1.5

Identificador:
http://www.w3.org/2001/04/xmlenc#rsa-1_5

El algoritmo RSAES-PKCS1-v1_5, especificado en RFC 3447 [PKCS1], no toma parámetros explícitos. Un ejemplo de un `EncryptionMethod`elemento RSA versión 1.5 es:

EJEMPLO 27

```
< Algoritmo del método de cifrado = "http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
```

La `CipherValue`clave para dicha clave cifrada es la codificación base64 [RFC2045] de la cadena de octetos calculada según RFC 3447 [PKCS1], sección 7.2.1: Operación de cifrado]. Como se especifica en la función EME-PKCS1-v1_5 RFC 3447 [PKCS1], sección 7.2.1, el valor ingresado a la función de transporte de claves es el siguiente:

EJEMPLO 28

```
CRIPTAR ( PAD ( TECLA ))
```

donde el relleno tiene la siguiente forma especial:

EJEMPLO 29

```
02 | PD * | 00 | llave
```

donde "|" es concatenación, "02" y "00" son octetos fijos del valor hexadecimal correspondiente, PS es una cadena de octetos pseudoaleatorios fuertes [ALEATORIO] de al menos ocho octetos de longitud, que no contienen octetos cero y lo suficientemente larga como para que el valor de la cantidad que se CRIPTA es un octeto más corta que el módulo RSA y "clave" es la clave que se transporta. La clave es de 192 bits para TRIPLEDES y de 128, 192 o 256 bits para AES.

Las implementaciones DEBEN admitir este algoritmo de transporte de claves para transportar claves TRIPLEDES de 192 bits. El soporte de este algoritmo para transportar otras claves es OPCIONAL . SE RECOMIENDA RSA-OAEP para el transporte de claves AES.

La cadena base64 [RFC2045] resultante es el valor del nodo de texto secundario del `CipherData`elemento, por ejemplo

EJEMPLO 30

```
<Datos de cifrado> <Valor de cifrado> IWijxQjUrcXBYoCei4QxjWo9Kg8D3p9tlWoT4

t0/gyTE96639In0FZFY2/rvP+/bMJ01EArmKZsR5VW3rwoPxw= </CipherValue> </CipherData>
```

(Nota: se agregó una nueva línea para que `CipherValue` quepa en esta página, pero no es parte del valor).

NOTA: NO SE RECOMIENDA la implementación de RSA v1.5 debido a los riesgos de seguridad asociados con el algoritmo.

5.5.2 RSA-OAEP

Identificador:
http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p (incluida `MGF1 with SHA1`la función de generación de máscara)
Identificador:
http://www.w3.org/2009/xmlenc11#rsa-oaep

El algoritmo RSAES-OAEP-ENCRYPT, como se especifica en RFC 3447 [PKCS1], tiene opciones que definen la función de resumen del mensaje y la función de generación de máscara, así como un `PSourceAlgorithm` parámetro opcional. Los valores predeterminados definidos en RFC 3447 son `SHA1` para el resumen del mensaje y `MGF1 with SHA1`para la función de generación de máscara. Tanto las funciones de resumen de mensajes como de generación de máscaras se utilizan en la operación EME-OAEP-ENCODE como parte de RSAES-OAEP-ENCRYPT.

El identificador http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p define la función de generación de máscara como el valor fijo de `MGF1 with SHA1`. En este caso NO DEBE proporcionarse el `xenc11:MGF`elemento opcional del `xenc:EncryptionMethod`elemento .

El identificador http://www.w3.org/2009/xmlenc11#rsa-oaep define la función de generación de máscara utilizando el `xenc11:MGF` elemento opcional del `xenc:EncryptionMethod`elemento. Si no está presente, `MGF1 with SHA1`se utilizará el valor predeterminado de.

Los siguientes URI definen los distintos valores de URI de la función de generación de máscaras que se pueden utilizar. Estos corresponden a los identificadores de objetos definidos en RFC 4055 [RFC4055]:

- MGF1 con SHA1: http://www.w3.org/2009/xmlenc11#mgf1sha1
- MGF1 con SHA224: http://www.w3.org/2009/xmlenc11#mgf1sha224
- MGF1 con SHA256: http://www.w3.org/2009/xmlenc11#mgf1sha256
- MGF1 con SHA384: http://www.w3.org/2009/xmlenc11#mgf1sha384
- MGF1 con SHA512: http://www.w3.org/2009/xmlenc11#mgf1sha512

De lo contrario, los dos identificadores definen el mismo uso del algoritmo RSA-OAEP, como sigue.

La función de resumen del mensaje DEBE especificarse utilizando el atributo Algoritmo del `ds:DigestMethod`elemento secundario del `xenc:EncryptionMethod`elemento. Si no se especifica, `SHA1`se utilizará el valor predeterminado de.

`PSourceAlgorithm`El valor del parámetro RSA-OAEP opcional PUEDE proporcionarse explícitamente colocando los octetos codificados en base64 en el `xenc:OAEPparams`elemento XML.

La definición y descripción del esquema XML Encryption 1.0 para el `EncryptionMethod`elemento se encuentran en la sección 3.2 El elemento EncryptionMethod . A continuación se muestra la adición de XML Encryption 1.1 para el tipo MGF:

```
Definición del esquema :

< nombre del elemento = "MGF" tipo = "xenc11:MGFType" />

<complexType name="MGFType">
  <complexContent>
```

```
      <restriction base="xenc11:AlgorithmIdentifierType">
        <attribute name="Algorithm" type="anyURI" use="required" />
      </restriction>
    </complexContent>
  </complexType>
```

An example of an RSA-OAEP element is:

EXAMPLE 31

```
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
  <OAEPparams>9lWu3Q==</OAEPparams>
  <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<EncryptionMethod>
```

EXAMPLE 32

```
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p">
            <OAEPparams>9lWu3Q==</OAEPparams>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <EncryptionMethod>
```

Another example is:

EXAMPLE 33

```
<EncryptionMethod Algorithm="http://www.w3.org/2009/xmlenc11#rsa-oaep">
  <OAEPparams>9lWu3Q==</OAEPparams>
  <xenc11:MGF Algorithm="http://www.w3.org/2001/04/xmlenc#MGF1withSHA1" />
  <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
<EncryptionMethod>
```

The `CipherValue` for an RSA-OAEP encrypted key is the base64 [RFC2045] encoding of the octet string computed as per RFC 3447 [PKCS1], section 7.1.1: Encryption operation. As described in the EME-OAEP-ENCODE function RFC 3447 [PKCS1], section 7.1.1, the value input to the key transport function is calculated using the message digest function and string specified in the `DigestMethod` and `OAEPparams` elements and using either the mask generator function specified with the `xenc11:MGF` element or the default `MGF1 with SHA1` specified in RFC 3447. The desired output length for EME-OAEP-ENCODE is one byte shorter than the RSA modulus.

The transported key size is 192 bits for TRIPLEDES and 128, 192, or 256 bits for AES. Implementations MUST implement RSA-OAEP for the transport of all key types and sizes that are mandatory to implement for symmetric encryption. They MAY implement RSA-OAEP for the transport of other keys.

5.6 Key Agreement

A Key Agreement algorithm provides for the derivation of a shared secret key based on a shared secret computed from certain types of compatible public keys from both the sender and the recipient. Information from the originator to determine the secret is indicated by an optional `OriginatorKeyInfo` parameter child of an `AgreementMethod` element while that associated with the recipient is indicated by an optional `RecipientKeyInfo`. A shared key is derived from this shared secret by a method determined by the Key Agreement algorithm.

Note: XML Encryption does not provide an online key agreement negotiation protocol. The `AgreementMethod` element can be used by the originator to identify the keys and computational procedure that were used to obtain a shared encryption key. The method used to obtain or select the keys or algorithm used for the agreement computation is beyond the scope of this specification.

The `AgreementMethod` element appears as the content of a `ds:KeyInfo` since, like other `ds:KeyInfo` children, it yields a key. This `ds:KeyInfo` is in turn a child of an `EncryptedData` or `EncryptedKey` element. The `Algorithm` attribute and `KeySize` child of the `EncryptionMethod` element under this `EncryptedData` or `EncryptedKey` element are implicit parameters to the key agreement computation. In cases where this `EncryptionMethod` algorithm URI is insufficient to determine the key length, a `KeySize` MUST have been included.

Key derivation algorithms (with associated parameters) may be explicitly declared by using the `xenc11:KeyDerivationMethod` element. This element will then be placed at the extensibility point of the `xenc:AgreementMethodType` (see below).

In addition, the sender may place a `KA-Nonce` element under `AgreementMethod` to assure that different keying material is generated even for repeated agreements using the same sender and recipient public keys. For example:

EXAMPLE 34

```
<EncryptedData>
  <EncryptionMethod Algorithm="Example:Block/Alg">
    <KeySize>80</KeySize>
  </EncryptionMethod>
  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <AgreementMethod Algorithm="example:Agreement/Algorithm">
      <KA-Nonce>Zm9v</KA-Nonce>
      <xenc11:KeyDerivationMethod
          Algorithm="http://www.w3.org/2009/xmlenc11#ConcatKDF">
        <xenc11:ConcatKDFParams
            AlgorithmID="00" PartyUInfo="" PartyVInfo="">
          <ds:DigestMethod
              Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
        </xenc11:ConcatKDFParams>
      </xenc11:KeyDerivationMethod>

      <OriginatorKeyInfo>
        <ds:KeyValue>....</ds:KeyValue>
      </OriginatorKeyInfo>
      <RecipientKeyInfo>
        <ds:KeyValue>....</ds:KeyValue>
      </RecipientKeyInfo>
    </AgreementMethod>
```

```
    </ds:KeyInfo>
    <CipherData>...</CipherData>
  </EncryptedData>
```

If the agreed key is being used to wrap a key, rather than data as above, then `AgreementMethod` would appear inside a `ds:KeyInfo` inside an `EncryptedKey` element.

The Schema for `AgreementMethod` is as follows:

```
Schema Definition:

<element name="AgreementMethod" type="xenc:AgreementMethodType"/>

<complexType name="AgreementMethodType" mixed="true">
  <sequence>
    <element name="KA-Nonce" minOccurs="0" type="base64Binary"/>
    <!-- <element ref="ds:DigestMethod" minOccurs="0"/> -->
    <any namespace="##other" minOccurs="0" maxOccurs="unbounded"/>
    <element name="OriginatorKeyInfo" minOccurs="0"
             type="ds:KeyInfoType"/>
    <element name="RecipientKeyInfo" minOccurs="0"
             type="ds:KeyInfoType"/>
  </sequence>
  <attribute name="Algorithm" type="anyURI" use="required"/>
</complexType>
```

5.6.1 Diffie-Hellman Key Values

Identifier:
http://www.w3.org/2001/04/xmlenc#DHKeyValue

Diffie-Hellman keys can appear directly within `KeyValue` elements or be obtained by `ds:RetrievalMethod` fetches as well as appearing in certificates and the like. The above identifier can be used as the value of the `Type` attribute of `Reference` or `ds:RetrievalMethod` elements.

As specified in [ESDH], a DH public key consists of up to six quantities, two large primes p and q, a "generator" g, the public key, and validation parameters "seed" and "pgenCounter". These relate as follows: The public key = (g**x mod p) where x is the corresponding private key; p = j*q + 1 where j >= 2. "seed" and "pgenCounter" are optional and can be used to determine if the Diffie-Hellman key has been generated in conformance with the algorithm specified in [ESDH]. Because the primes and generator can be safely shared over many DH keys, they may be known from the application environment and are optional. The schema for a `DHKeyValue` is as follows:

```
Schema Definition:

<element name="DHKeyValue" type="xenc:DHKeyValueType"/>

<complexType name="DHKeyValueType">
  <sequence>
    <sequence minOccurs="0">
      <element name="P" type="ds:CryptoBinary"/>
      <element name="Q" type="ds:CryptoBinary"/>
      <element name="Generator"type="ds:CryptoBinary"/>
    </sequence>
    <element name="Public" type="ds:CryptoBinary"/>
    <sequence minOccurs="0">
      <element name="seed" type="ds:CryptoBinary"/>
      <element name="pgenCounter" type="ds:CryptoBinary"/>
    </sequence>
  </sequence>
</complexType>
```

5.6.2 Diffie-Hellman Key Agreement

The Diffie-Hellman (DH) key agreement protocol [ESDH] involves the derivation of shared secret information based on compatible DH keys from the sender and recipient. Two DH public keys are compatible if they have the same prime and generator. If, for the second one, `Y = g**y mod p`, then the two parties can calculate the shared secret `ZZ = ( g**(x*y) mod p )` even though each knows only their own private key and the other party's public key. Leading zero bytes MUST be maintained in `ZZ` so it will be the same length, in bytes, as `p`. The size of `p` MUST be at least 512 bits and `g` at least 160 bits. There are numerous other complex security considerations in the selection of `g`, `p`, and a random `x` as described in [ESDH].

The Diffie-Hellman shared secret `zz` is used as the input to a KDF to produce a secret key. XML Signature 1.0 defined a specific KDF to be used with Diffie-Hellman; that KDF is now known as the "Legacy KDF" and is defined in Section 5.6.2.2. Use of Diffie-Hellman with explicit KDFs is described in Section 5.6.2.1.

Implementation of Diffie-Hellman key agreement is OPTIONAL. However, if implemented, such implementations MUST support the Legacy Key Derivation Function and SHOULD support Diffie-Hellman with explicit Key Derivation Functions

An example of a DH `AgreementMethod` element using the Legacy Key Derivation Function (Section 5.6.2.2) is as follows:

EXAMPLE 35

```
  <AgreementMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#dh"
      ds:xmlns="http://www.w3.org/2000/09/xmldsig#">
    <KA-Nonce>Zm9v</KA-Nonce>
    <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
    <OriginatorKeyInfo>
      <ds:X509Data>
        <ds:X509Certificate>...</ds:X509Certificate>
      </ds:X509Data>
    </OriginatorKeyInfo>
    <RecipientKeyInfo>
      <ds:KeyValue>...</ds:KeyValue>
```

```
  </RecipientKeyInfo>
</AgreementMethod>
```

5.6.2.1 Diffie-Hellman Key Agreement with Explicit Key Derivation Functions

Identifier:
: http://www.w3.org/2009/xmlenc11#dh-es

It is RECOMMENDED that the shared key material for a Diffie-Hellman key agreement be calculated from the Diffie-Hellman shared secret using a key derivation function (KDF) in accordance with Section 5.4.

An example of a DH `AgreementMethod` element using an explicit key derivation function is as follows:

EXAMPLE 36

```
<xenc:AgreementMethod Algorithm="http://www.w3.org/2009/xmlenc11#dh-es">
  <xenc11:KeyDerivationMethod Algorithm="http://www.w3.org/2009/xmlenc11#ConcatKDF">
    <xenc11:ConcatKDFParams AlgorithmID="00" PartyUInfo="" PartyVInfo="">
      <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
    </xenc11:ConcatKDFParams>
  </xenc11:KeyDerivationMethod>
  <xenc:OriginatorKeyInfo>
    <ds:X509Data>
      <ds:X509Certificate><!-- X.509 Certificate here --></ds:X509Certificate>
    </ds:X509Data>
  </xenc:OriginatorKeyInfo>
  <xenc:RecipientKeyInfo>
    <ds:X509Data>
      <ds:X509SKI></ds:X509SKI>
      <!-- hint for the recipient's private key -->
    </ds:X509Data>
  </xenc:RecipientKeyInfo>
</xenc:AgreementMethod>
```

5.6.2.2 Diffie-Hellman Key Agreement with Legacy Key Derivation Function

Identifier:
: http://www.w3.org/2001/04/xmlenc#dh

XML Signature 1.0 defined a specific KDF for use with Diffie-Hellman key agreement. In order to guarantee interoperability, implementations that choose to implement Diffie-Hellman MUST support the use of the Diffie-Hellman Legacy KDF defined in this section.

Assume that the Diffie-Hellman shared secret is the octet sequence `ZZ`. The Diffie-Hellman Legacy KDF calculates the shared keying material as follows:

EXAMPLE 37

```
Keying Material = KM(1) | KM(2) | ...
```

where "|" is byte stream concatenation and

EXAMPLE 38

```
KM(counter) = DigestAlg ( ZZ | counter | EncryptionAlg |
            KA-Nonce | KeySize )
```

`DigestAlg`
: The message digest algorithm specified by the `DigestMethod` child of `AgreementMethod`.

`EncryptionAlg`
: The URI of the encryption algorithm, including possible key wrap algorithms, in which the derived keying material is to be used ("Example:Block/Alg" in the example above), not the URI of the agreement algorithm. This is the value of the `Algorithm` attribute of the `EncryptionMethod` child of the `EncryptedData` or `EncryptedKey` grandparent of `AgreementMethod`.

`KA-Nonce`
: The base64 decoding the content of the `KA-Nonce` child of `AgreementMethod`, if present. If the `KA-Nonce` element is absent, it is null.

`Counter`
: A one byte counter starting at one and incrementing by one. It is expressed as two hex digits where letters A through F are in upper case.

`KeySize`
: The size in bits of the key to be derived from the shared secret as the UTF-8 string for the corresponding decimal integer with only digits in the string and no leading zeros. For some algorithms the key size is inherent in the URI. For others, such as most stream ciphers, it must be explicitly provided.

For example, the initial `(KM(1))` calculation for the `EncryptionMethod` of the Key Agreement example (section 5.5) would be as follows, where the binary one byte counter value of 1 is represented by the two character UTF-8 sequence `01`, `ZZ` is the shared secret, and "`foo`" is the base64 decoding of "`Zm9v`".

EXAMPLE 39

```
SHA-1 ( ZZ01Example:Block/Algfoo80 )
```

Assuming that `ZZ` is `0xDEADBEEF`, that would be

EXAMPLE 40

```
SHA-1( 0xDEADBEEF30314578616D706C653A426C6F636B2F416C67666F6F3830 )
```

whose value is

EXAMPLE 41

```
0x534C9B8C4ABDCB50038B42015A181711068B08C1
```

Each application of `DigestAlg` for successive values of `Counter` will produce some additional number of bytes of keying material. From the concatenated string of one or more `KM`'s, enough leading bytes are taken to meet the need for an actual key and the remainder discarded. For example, if `DigestAlg` is SHA-1 which produces 20 octets of hash, then for 128 bit AES the first 16 bytes from `KM(1)` would be taken and the remaining 4 bytes discarded. For 256 bit AES, all of `KM(1)` suffixed with the first 12 bytes of KM(2) would be taken and the remaining 8 bytes of `KM(2)` discarded.

5.6.3 Elliptic Curve Diffie-Hellman (ECDH) Key Values

Identifier:
http://www.w3.org/2009/xmldsig11#ECKeyValue

ECDH has identical public key parameters as ECDSA and can be represented with the `ECKeyValue` element [XMLDSIG-CORE1]. Note that if the curve parameters are explicitly stated using the ECParameters element, then the Cofactor element MUST be included.

As with Diffie-Hellman keys, Elliptic Curve Key Values can appear directly within `KeyValue` elements or be obtained by `ds:RetrievalMethod` fetches as well as appearing in certificates and the like. The above identifier can be used as the value of the `Type` attribute of `Reference` or `ds:RetrievalMethod` elements.

5.6.4 Elliptic Curve Diffie-Hellman (ECDH) Key Agreement (Ephemeral-Static Mode)

Identifier:
http://www.w3.org/2009/xmlenc11#ECDH-ES

ECDH is the elliptic curve analogue to the Diffie-Hellman key agreement algorithm. Details of the ECDH primitive can be found in [ECC-ALGS]. When ECDH is used in Ephemeral-Static (ES) mode, the recipient has a static key pair, but the sender generates a ephemeral key pair for each message. The same ephemeral key may be used when there are multiple recipients that use the same curve parameters.

Compliant implementations are REQUIRED to support ECDH-ES key agreement using the P-256 prime curve specified in Section D.2.3 of FIPS 186-3 [FIPS-186-3]. (This is the same curve that is REQUIRED in XML Signature 1.1 to be supported for the ECDSAwithSHA256 algorithm.) It is further RECOMMENDED that implementations also support the P-384 and P-521 prime curves for ECDH-ES; these curves are defined in Sections D.2.4 and D.2.5 of FIPS 186-3, respectively.

The shared key material is calculated from the Diffie-Hellman shared secret using a key derivation function (KDF). While applications may define other KDFs, compliant implementations MUST implement ConcatKDF (see section 5.4.1 ConcatKDF). An example of `xenc:EncryptedData` using the ECDH-ES key agreement algorithm with the ConcatKDF key derivation algorithm is as follows:

EXAMPLE 42

```
<xenc:EncryptedData
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:dsig11="http://www.w3.org/2009/xmldsig11#"
    xmlns:xenc11="http://www.w3.org/2009/xmlenc11#"
    Type="http://www.w3.org/2001/04/xmlenc#">

  <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc" />
  <!-- describes the encrypted AES content encryption key -->
  <ds:KeyInfo>
    <xenc:EncryptedKey>
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
      <!-- describes the key encryption key -->
      <ds:KeyInfo>
        <xenc:AgreementMethod Algorithm="http://www.w3.org/2009/xmlenc11#ECDH-ES">
          <xenc11:KeyDerivationMethod Algorithm="http://www.w3.org/2009/xmlenc11#ConcatKDF">
            <xenc11:ConcatKDFParams AlgorithmID="00" PartyUInfo="" PartyVInfo="">
              <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            </xenc11:ConcatKDFParams>
          </xenc11:KeyDerivationMethod>
          <xenc:OriginatorKeyInfo>
            <ds:KeyValue>
              <dsig11:ECKeyValue>
                <!-- ephemeral ECC public key of the originator -->
              </dsig11:ECKeyValue>
            </ds:KeyValue>
          </xenc:OriginatorKeyInfo>
          <xenc:RecipientKeyInfo>
            <ds:X509Data>
              <ds:X509SKI></ds:X509SKI>
              <!-- hint for the recipient's private key -->
            </ds:X509Data>
          </xenc:RecipientKeyInfo>
        </xenc:AgreementMethod>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue><!-- encrypted AES content encryption key --></xenc:CipherValue>
      </xenc:CipherData>
    </xenc:EncryptedKey>
  </ds:KeyInfo>
```

```
    <xenc:CipherData>
      <xenc:CipherValue>
        <!-- encrypted data -->
      </xenc:CipherValue>
    </xenc:CipherData>

  </xenc:EncryptedData>
```

5.7 Symmetric Key Wrap

Symmetric Key Wrap algorithms are shared secret key encryption algorithms especially specified for encrypting and decrypting symmetric keys. When wrapped keys are used, then an `EncryptedKey` element will appear as a child of a `ds:KeyInfo` element. This `EncryptedKey` element will have an `EncryptionMethod` child whose `Algorithm` attribute in turn identifies the key wrap algorithm.

The algorithm for which the encrypted key is intended depends on the context of the `ds:KeyInfo` element: `ds:KeyInfo` can occur as a child of either an `EncryptedData` or `EncryptedKey` element; in both cases, `ds:KeyInfo` will have an `EncryptionMethod` sibling that identifies the algorithm.

EXAMPLE 43

```
  <EncryptedData |EncryptedKey>
    <EncryptionMethod Algorithm="@alg1"/>
    <ds:KeyInfo>
      <EncryptedKey>
        <EncryptionMethod Algorithm="@alg2"/>
      </EncryptedKey>
    </ds:KeyInfo>
  </EncryptedData |EncryptedKey>
```

5.7.1 CMS Triple DES Key Wrap

Identifiers:
    http://www.w3.org/2001/04/xmlenc#kw-tripledes

XML Encryption implementations MUST support TRIPLEDES wrapping of 168 bit keys as described in [CMS-WRAP] and may optionally support TRIPLEDES wrapping of other keys.

An example of a TRIPLEDES Key Wrap `EncryptionMethod` element is as follows:

EXAMPLE 44

```
  <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-tripledes" />
```

5.7.2 AES KeyWrap

Identifiers:
    http://www.w3.org/2001/04/xmlenc#kw-aes128
    http://www.w3.org/2001/04/xmlenc#kw-aes192
    http://www.w3.org/2001/04/xmlenc#kw-aes256

Implementation of AES key wrap is described in [AES-WRAP]. It provides for confidentiality and integrity. This algorithm is defined only for inputs which are a multiple of 64 bits. The information wrapped need not actually be a key. The algorithm is the same whatever the size of the AES key used in wrapping, called the key encrypting key or KEK. The implementation requirements are indicated below.

128 bit AES Key Encrypting Key
    Implementation of wrapping 128 bit keys REQUIRED.
    Wrapping of other key sizes OPTIONAL.
192 bit AES Key Encrypting Key
    All support OPTIONAL.
256 bit AES Key Encrypting Key
    Implementation of wrapping 256 bit keys REQUIRED.
    Wrapping of other key sizes OPTIONAL.

5.8 Message Digest

Message digest algorithms can be used in `AgreementMethod` as part of the key derivation, within RSA-OAEP encryption as a hash function, and in connection with the HMAC message authentication code method [HMAC] as described in [XMLDSIG-CORE1].) Use of SHA-256 is strongly recommended over SHA-1 because recent advances in cryptanalysis (see e.g. [SHA-1-Analysis], [SHA-1-Collisions]) have cast doubt on the long-term collision resistance of SHA-1. Therefore, SHA-1 support is REQUIRED in this specification only for backwards-compatibility reasons.

5.8.1 SHA1

Identifier:
    http://www.w3.org/2000/09/xmldsig#sha1

The SHA-1 algorithm [FIPS-180-3] takes no explicit parameters. An example of an SHA-1 `DigestMethod` element is:

EXAMPLE 45

```
  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
```

A SHA-1 digest is a 160-bit string. The content of the `DigestValue` element shall be the base64 encoding of this bit string viewed as a 20-octet octet stream. For example, the `DigestValue` element for the message digest:

EXAMPLE 46

```
A9993E36 4706816A BA3E2571 7850C26C 9CD0D89D
```

from Appendix A of the SHA-1 standard would be:

EXAMPLE 47

```
<DigestValue>qZk+NkcGgWq6PiVxeFDCbJzQ2J0=</DigestValue>
```

5.8.2 SHA256

Identifier:
http://www.w3.org/2001/04/xmlenc#sha256

The SHA-256 algorithm [FIPS-180-3] takes no explicit parameters. An example of an SHA-256 `DigestMethod` element is:

EXAMPLE 48

```
<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256" />
```

A SHA-256 digest is a 256-bit string. The content of the `DigestValue` element shall be the base64 encoding of this bit string viewed as a 32-octet octet stream.

5.8.3 SHA384

Identifier:
http://www.w3.org/2001/04/xmlenc#sha384

The SHA-384 algorithm [FIPS-180-3] takes no explicit parameters. An example of an SHA-384 `DigestMethod` element is:

EXAMPLE 49

```
<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha384" />
```

A SHA-384 digest is a 384-bit string. The content of the `DigestValue` element shall be the base64 encoding of this bit string viewed as a 48-octet octet stream.

5.8.4 SHA512

Identifier:
http://www.w3.org/2001/04/xmlenc#sha512

The SHA-512 algorithm [FIPS-180-3] takes no explicit parameters. An example of an SHA-512 `DigestMethod` element is:

EXAMPLE 50

```
<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha512" />
```

A SHA-512 digest is a 512-bit string. The content of the `DigestValue` element shall be the base64 encoding of this bit string viewed as a 64-octet octet stream.

5.8.5 RIPEMD-160

Identifier:
http://www.w3.org/2001/04/xmlenc#ripemd160

The RIPEMD-160 algorithm [RIPEMD-160] takes no explicit parameters. An example of an RIPEMD-160 `DigestMethod` element is:

EXAMPLE 51

```
<DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#ripemd160" />
```

A RIPEMD-160 digest is a 160-bit string. The content of the `DigestValue` element shall be the base64 encoding of this bit string viewed as a 20-octet octet stream.

5.9 Canonicalization

A Canonicalization of XML is a method of consistently serializing XML into an octet stream as is necessary prior to encrypting XML.

5.9.1 Inclusive Canonicalization

Identifiers:
http://www.w3.org/TR/2001/REC-xml-c14n-20010315

http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments
http://www.w3.org/2006/12/xml-c14n11
http://www.w3.org/2006/12/xml-c14n11#WithComments

Canonical XML [XML-C14N11] is a method of serializing XML which includes the in scope namespace and xml namespace attribute context from ancestors of the XML being serialized.

If XML is to be encrypted and then later decrypted into a different environment and it is desired to preserve namespace prefix bindings and the value of attributes in the "xml" namespace of its original environment, then the canonical XML with comments version of the XML should be the serialization that is encrypted.

5.9.2 Exclusive Canonicalization

Identifiers:
http://www.w3.org/2001/10/xml-exc-c14n#
http://www.w3.org/2001/10/xml-exc-c14n#WithComments

Exclusive XML Canonicalization [XML-EXC-C14N] serializes XML in such a way as to include to the minimum extent practical the namespace prefix binding and xml namespace attribute context inherited from ancestor elements.

It is the recommended method where the outer context of a fragment which was signed and then encrypted may be changed. Otherwise the validation of the signature over the fragment may fail because the canonicalization by signature validation may include unnecessary namespaces into the fragment.

6. Security Considerations

6.1 Chosen-Ciphertext Attacks

A number of chosen-ciphertext attacks against implementations of this specification have been published and demonstrated. They all involve the following elements:

1. The attacker knows about the format of the cleartext.
2. The attacker is able to submit substantial numbers of ciphertext messages.
3. The attacker is able to send arbitrary ciphertext, based on previous results.
4. The attacker is able to force the server to use the same key (secret key by CBC-based attacks and server's private key by PKCS#1.5 attacks) for processing of the adapted ciphertext.
5. The server attempting to decrypt the ciphertext in some way signals whether the decrypted text is well-formed or not.

The attacker uses the knowledge of the format and the information about well-formedness to construct a series of ciphertext guesses which reveal the plaintext with much less work than brute force. Attacks of this type have been demonstrated against symmetric encryption using CBC mode [XMLENC-CBC-ATTACK][XMLENC-CBC-ATTACK-COUNTERMEASURES] and on PKCS#1 v1.5. Other future attacks can be expected whenever these conditions are met.

6.1.1 Attacks against the encrypted data (`<EncryptedData>` part)

Using the CBC-based chosen-ciphertext attacks, the attacker sends to the server an XML document with modified encrypted data in the symmetric part (`<EncryptedData>`). After a few requests, the attacker is able to get the whole cleartext without knowledge of the symmetric key.

It would seem that these attacks can be countered by by disrupting any of the conditions, however in practice only preventing condition 3 (sending arbitrary ciphertext) is fully effective. To counter condition 3, it is necessary for the decrypting system to require authenticated integrity protection over the ciphertext. However, unless the mechanism used is bound to the encryption key, there will no way to be sure that the signer is not attempting to recover the plaintext. The simplest and most efficient way to do this is to use an authenticating block mode, such as GCM. An alternative would be an HMAC based on the encryption key over the ciphertext, but it is less efficient and provides no advantages.

Other countermeasures are not likely to be effective. Limiting the number of messages presented or the number of messages using the same key is not practical in large server farms. Attackers can spread their attempts over different servers and long or short periods of time, to foil attempts to detect attacks in progress or determine the location of the attacker.

Signaling well-formedness can occur by emitting different messages for distinct security errors or by exhibiting timing differences. Implementations should avoid these practices, however that is not sufficient to prevent such attacks in an XML protocol environment, such as SOAP. Using a technique called encryption wrapping, the attacker can insert the ciphertext in some schema-legal part of the message. If the decryption code notices a format error, an error will be returned, but if not the message will be passed to the application which will ignore the bogus plaintext and ultimately respond with an application level success or failure message.

6.1.2 Attacks against the encrypted key (Bleichenbacher's Million question attack on PKCS#1.5)

The goal of the attacker applying the Bleichenbacher's attack is to get the symmetric secret key, which is encrypted in the `<EncryptedKey>` part. Afterward, he would be able to decrypt the whole data carried in the `<EncryptedData>` part.

The basic idea of this attack is to modify the data in the `<EncryptedKey>` part, send the document to the server, and observe if the modified ciphertext contains PKCS#1.5 conformant data. This can be done by:

1. Observing fault messages of the server notifying directly that the request was not PKCS#1.5 conformant (this should not happen).
2. Enlarging the data in the `<EncryptedData>` part and observing the timing differences between inclusion of PKCS-valid and PKCS-invalid keys: if the key is PKCS-valid, the session key is extracted, and the large data is decrypted. Otherwise, the session key cannot be extracted and the large data is not processed, which yields a timing difference.
3. Making specific modifications of the `<EncryptedData>` part based on CBC and padding-properties.

These problems are described in detail in RFC 3218 [RFC3218].

The most effective countermeasure against the timing attack (2) is to generate a random secret key every time when the decrypted data was not PKCS#1-conformant. This way, the attacker would not get any timing side-channel.

Please note however that this is not a valid countermeasure against the specific modification of the <EncryptedData> described in part (3). The attacker could still use a few millions of requests to decrypt the encrypted symmetric key. Therefore, we recommend the usage of RSA-OAEP. RSA-OAEP also has a risk of a chosen ciphertext attack [OAEP-ATTACK] which can be mitigated in security library implementations.

6.1.3 Backwards Compatibility Attacks

Use of state-of-the-art and secure encryption algorithms such as RSA-OAEP and AES-GCM can become insecure when the adversary can force the server to process eavesdropped ciphertext with legacy algorithms such as RSA-PKCS#1 v1.5 or AES-CBC [XMLENC-BACKWARDS-COMP]:

1. The attacker may be able to break the security of an AES-GCM ciphertext if he is able to force the server to process the ciphertext with AES-CBC and the same symmetric key.
2. The attacker may be able to decrypt an RSA-OAEP ciphertext if he is able to force the server to process the ciphertext with RSA-PKCS#1 v1.5 and the same asymmetric key.
3. The attacker may be able to forge valid server signatures if the server decrypts RSA-PKCS#1 v1.5 ciphertexts and the signatures are computed with the same asymmetric key pair.

Accordingly, in situations where an attacker may be able to mount chosen-ciphertext attacks, we recommend the following to implementers:

1. Implementations SHOULD always use a different public key pair for data confidentiality and for data integrity functionality.
2. Implementations using symmetric keys SHOULD NOT use the same key material for different algorithms, even if serving the same purpose. Key derivation based on a single key and the algorithm identifier can be used to accomplish this, for example.
3. Implementations that plan to use the same symmetric key for both confidentiality and integrity functions SHOULD use it as the basis for a key derivation producing different keys for those functions.
4. Implementations SHOULD restrict algorithm usage to algorithms known to be secure in the face of chosen-ciphertext attacks (RSA-OAEP, AES-GCM). In that case, documents containing RSA-PKCS#1 v1.5 [XMLENC-PKCS15-ATTACK] and AES-CBC [XMLENC-CBC-ATTACK] ciphertexts SHOULD be rejected without decryption.

6.2 Relationship to XML Digital Signatures

The application of both encryption and digital signatures over portions of an XML document can make subsequent decryption and signature verification difficult. In particular, when verifying a signature one must know whether the signature was computed over the encrypted or unencrypted form of elements.

A separate, but important, issue is introducing cryptographic vulnerabilities when combining digital signatures and encryption over a common XML element. Hal Finney has suggested that encrypting digitally signed data, while leaving the digital signature in the clear, may allow plaintext guessing attacks. This vulnerability can be mitigated by using secure hashes and the nonces in the text being processed.

In accordance with the requirements document [XML-ENCRYPTION-REQ] the interaction of encryption and signing is an application issue and out of scope of the specification. However, we make the following recommendations:

1. When data is encrypted, any digest or signature over that data should be encrypted. This satisfies the first issue in that only those signatures that can be seen can be validated. It also addresses the possibility of a plaintext guessing vulnerability, though it may not be possible to identify (or even know of) all the signatures over a given piece of data.

2. Employ the "decrypt-except" signature transform [XMLENC-DECRYPT]. It works as follows: during signature transform processing, if you encounter a decrypt transform, decrypt all encrypted content in the document except for those excepted by an enumerated set of references.

Additionally, while the following warnings pertain to incorrect inferences by the user about the authenticity of information encrypted, applications should discourage user misapprehension by communicating clearly which information has integrity, or is authenticated, confidential, or non-repudiable when multiple processes (e.g., signature and encryption) and algorithms (e.g., symmetric and asymmetric) are used:

1. When an encrypted envelope contains a signature, the signature does not necessarily protect the authenticity or integrity of the ciphertext [Davis].

2. While the signature secures plaintext it only covers that which is signed, recipients of encrypted messages must not infer integrity or authenticity of other unsigned information (e.g., headers) within the encrypted envelope, see [XMLDSIG-CORE1], section 8.1.1 Only What is Signed is Secure].

6.3 Information Revealed

Where a symmetric key is shared amongst multiple recipients, that symmetric key should *only* be used for the data intended for *all* recipients; even if one recipient is not directed to information intended (exclusively) for another in the same symmetric key, the information might be discovered and decrypted.

Additionally, application designers should be careful not to reveal any information in parameters or algorithm identifiers (e.g., information in a URI) that weakens the encryption.

6.4 Nonce and IV (Initialization Value or Vector)

An undesirable characteristic of many encryption algorithms and/or their modes is that the same plaintext when encrypted with the same key has the same resulting ciphertext. While this is unsurprising, it invites various attacks which are mitigated by including an arbitrary and non-repeating (under a given key) data with the plaintext prior to encryption. In encryption chaining modes this data is the first to be encrypted and is consequently called the IV (initialization value or vector).

Different algorithms and modes have further requirements on the characteristic of this information (e.g., randomness and secrecy) that affect the features (e.g., confidentiality and integrity) and their resistance to attack.

Given that XML data is redundant (e.g., Unicode encodings and repeated tags) and that attackers may know the data's structure (e.g., DTDs and schemas) encryption algorithms must be carefully implemented and used in this regard.

For the Cipher Block Chaining (CBC) mode used by this specification, the IV must not be reused for any key and should be random, but it need not be secret. Additionally, under this mode an adversary modifying the IV can make a known change in the plain text after decryption. This attack can be avoided by securing the integrity of the plain text data, for example by signing it.

Note: CBC block encryption algorithms should not be used without consideration of possibly severe security risks.

For the Galois/Counter Mode (GCM) used by this specification, the IV must not be reused for any key and should be random, but it need not be secret.

6.5 Denial of Service

This specification permits recursive processing. For example, the following scenario is possible: `EncryptedKey` **A** requires `EncryptedKey` **B** to be decrypted, which itself requires `EncryptedKey` **A**! Or, an attacker might submit an `EncryptedData` for decryption that references network resources that are very large or continually redirected. Consequently, implementations should be able to restrict arbitrary recursion and the total amount of processing and networking resources a request can consume.

6.6 Unsafe Content

XML Encryption can be used to obscure, via encryption, content that applications (e.g., firewalls, virus detectors, etc.) consider unsafe (e.g., executable code, viruses, etc.). Consequently, such applications must consider encrypted content to be as unsafe as the unsafest content transported in its application context. Consequently, such applications may choose to (1) disallow such content, (2) require access to the decrypted form for inspection, or (3) ensure that arbitrary content can be safely processed by receiving applications.

6.7 Error Messages

Implementations SHOULD NOT provide detailed error responses related to security algorithm processing. Error messages should be limited to a generic error message to avoid providing information to a potential attacker related to the specifics of the algorithm implementation. For example, if an error occurs in decryption processing the error response should be a generic message providing no specifics on the details of the processing error.

6.8 Timing Attacks

It has been known for some time that it is feasible for an attacker to recover keys or cleartext by repeatedly sending chosen ciphertext and measuring the time required to process different requests with different types of errors. It has been demonstrated that attacks of this type are practical even when communicating over large and busy networks, especially if the receiver is willing to process large numbers of ciphertext blocks.

Implementers SHOULD ensure that distinct errors detected during security algorithm processing do not consume systematically different amounts of processing time from each other. Implementers SHOULD consult the technical literature for more details on specific attacks and recommended countermeasures.

Deployments SHOULD treat as suspect inputs when a large number of security algorithm processing errors are detected within a short period of time, especially in messages from the same origin.

6.9 CBC Block Encryption Vulnerability

Note: CBC block encryption algorithms should not be used without consideration of possibly severe security risks.

7. Conformance

An implementation is conformant to this specification if it successfully generates syntax according to the schema definitions and satisfies all MUST/REQUIRED/SHALL requirements, including algorithm support and processing. Processing requirements are specified over the roles of **decryptor**, **encryptor**, and their calling **application**.

8. XML Encryption Media Type

8.1 Introduction

XML Encryption Syntax and Processing (XMLENC-CORE1, this document) specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML Encryption element which contains or references the cipher data.

The `application/xenc+xml` media type allows XML Encryption applications to identify encrypted documents. Additionally it allows applications cognizant of this media-type (even if they are not XML Encryption implementations) to note that the media type of the decrypted (original) object might be a type other than XML.

8.2 application/xenc+xml Registration

This is a media type registration as defined in Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures [MIME-REG]

Type name: application

Subtype name: xenc+xml

Required parameters: none

Optional parameters: charset

The allowable and recommended values for, and interpretation of the charset parameter are identical to those given for 'application/xml' in section 3.2 of RFC 3023 [XML-MT].

Encoding considerations:

The encoding considerations are identical to those given for 'application/xml' in section 3.2 of RFC 3023 [XML-MT].

Security considerations:

See the (XMLENC-CORE1, this document) Security Considerations section.

Interoperability considerations: none

Published specification: (XMLENC-CORE1, this document)

Applications which use this media type:

XML Encryption is device-, platform-, and vendor-neutral and is supported by a range of Web applications.

Additional Information:

Magic number(s): none

Although no byte sequences can be counted on to consistently identify XML Encryption documents, there will be XML documents in which the root element's `QName`'s `LocalPart` is `'EncryptedData'` or '`EncryptedKey`' with an associated namespace name of 'http://www.w3.org/2001/04/xmlenc#'. The `application/xenc+xml` type name MUST only be used for data objects in which the root element is from the XML Encryption namespace. XML documents which contain these element types in places other than the root element can be described using facilities such as [XMLSCHEMA-1], [XMLSCHEMA-2].

File extension(s): .xml

Macintosh File Type Code(s): "TEXT"

Person & email address to contact for further information:

World Wide Web Consortium <web-human at w3.org>

Intended usage: COMMON

Author/Change controller:

The XML Encryption specification is a work product of the World Wide Web Consortium (W3C) which has change control over the specification.

9. Schema

9.1 XSD Schema

XML Encryption Core Schema Instance
xenc-schema.xsd
XML Encryption 1.1 Schema Instance
xenc-schema11.xsd
This schema document defines the additional material defined in XML Encryption 1.1.
Example (non-normative)
enc-example.xml (not cryptographically valid but exercises much of the schema)

9.2 RNG Schema

This section is non-normative.

Non-normative RELAX NG schema [RELAXNG-SCHEMA] information is available in a separate document [XMLSEC-RELAXNG].

A. Reserved Algorithm Identifiers

This informative section outlines the definition and reserves identifiers for algorithms that have no requirements for implementation and have not been tested for interoperability.

A.1 AES KeyWrap with Padding

This section is non-normative.

Identifiers:
http://www.w3.org/2009/xmlenc11#kw-aes-128-pad
http://www.w3.org/2009/xmlenc11#kw-aes-192-pad
http://www.w3.org/2009/xmlenc11#kw-aes-256-pad

These identifiers are reserved for symmetric key wrapping using the AES key wrap with padding algorithm with a 128, 192, and 256 bit AES key encrypting key, respectively. Implementation of AES key wrap with padding is defined in [AES-WRAP-PAD]. The algorithm is defined for inputs between 9 and 2^32 octets. Unlike the unpadded AES Key Wrap algorithm, the input length is not constrained to multiples of 64 bits (8 octets).

Note that the wrapped key will be distinct from the one generated by the unpadded AES Key Wrap algorithm, even if the input length is a multiple of 64 bits.

B. References

Dated references below are to the latest known or appropriate edition of the referenced work. The referenced works may be subject to revision, and conformant implementations may follow, and are encouraged to investigate the appropriateness of following, some or all more recent editions or replacements of the works cited. It is in each case implementation-defined which editions are supported.

B.1 Normative references

[AES]
: *NIST FIPS 197: Advanced Encryption Standard (AES)*. November 2001. URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[AES-WRAP]
: J. Schaad; R. Housley. *RFC3394: Advanced Encryption Standard (AES) Key Wrap Algorithm*. September 2002. IETF Informational RFC. URL: http://www.ietf.org/rfc/rfc3394.txt

[AES-WRAP-PAD]
: R. Housley; M. Dworkin. *RFC 5649: Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm*. August 2009. IETF Informational RFC. URL: http://www.ietf.org/rfc/rfc5649.txt

[ANSI-X9-44-2007]
: *ANSI X9.44-2007: Key Establishment Using Integer Factorization Cryptography.* URL: http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+X9.44-2007

[CMS-WRAP]
: R. Housley. *RFC3217: Triple-DES and R2 Key Wrapping*. December 2001. IETF Informational RFC. URL: http://www.ietf.org/rfc/rfc3217.txt

[DES]
: *NIST FIPS 46-3: Data Encryption Standard (DES)* . October 1999. URL: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

[ESDH]
: E. Rescorla. *Diffie-Hellman Key Agreement Method.*. IETF RFC 2631 Standards Track, 1999. URL: http://www.ietf.org/rfc/rfc2631.txt

[EXI]
: Takuki Kamiya; John Schneider. *Efficient XML Interchange (EXI) Format 1.0*. 8 December 2009. W3C Candidate Recommendation. URL: http://www.w3.org/TR/2009/CR-exi-20091208/

[FIPS-180-3]
: *FIPS PUB 180-3 Secure Hash Standard*. U.S. Department of Commerce/National Institute of Standards and Technology. URL: http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[FIPS-186-3]
: *FIPS PUB 186-3: Digital Signature Standard (DSS)*. June 2009. U.S. Department of Commerce/National Institute of Standards and Technology. URL: http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

[HMAC]
: H. Krawczyk, M. Bellare, R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. February 1997. IETF RFC 2104. URL: http://www.ietf.org/rfc/rfc2104.txt

[NFC]
: M. Davis, Ken Whistler. *TR15, Unicode Normalization Forms.*. 17 September 2010, URL: http://www.unicode.org/reports/tr15/

[PKCS1]
: J. Jonsson and B. Kaliski. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1.* RFC 3447 (Informational), February 2003. URL: http://www.ietf.org/rfc/rfc3447.txt

[PKCS5]
: B. Kaliski. *PKCS #5 v2.0: Password-Based Cryptography Standard*. September 2000. IETF RFC 2898. URL: http://www.ietf.org/rfc/rfc2898.txt

[PKCS5Amd1]
: *PKCS #5 v2.0 Amendment 1: XML Schema for Password-Based Cryptography* RSA Laboratories, March 2007. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs-5v2-0a1.pdf

[RANDOM]
: D. Eastlake, S. Crocker, J. Schiller. *Randomness Recommendations for Security.*. IETF RFC 4086. June 2005. URL: http://www.ietf.org/rfc/rfc4086.txt

[RFC2045]
: N. Freed and N. Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.* November 1996. URL: http://www.ietf.org/rfc/rfc2045.txt

[RFC2119]
: S. Bradner. *Key words for use in RFCs to Indicate Requirement Levels.* March 1997. Internet RFC 2119. URL: http://www.ietf.org/rfc/rfc2119.txt

[RFC4055]
: J. Schaad, B. Kaliski, R. Housley. *Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. June 2005. IETF RFC 4055. URL: http://www.ietf.org/rfc/rfc4055.txt

[RIPEMD-160]
: B. Preneel, A. Bosselaers, and H. Dobbertin. *The Cryptographic Hash Function RIPEMD-160*. CryptoBytes, Volume 3, Number 2. pp. 9-14, RSA Laboratories 1997. URL: http://www.cosic.esat.kuleuven.be/publications/article-317.pdf

[SP800-38D]
: M. Dworkin. *NIST Special Publication 800-38D: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. November 2007 URL: http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf

[SP800-56A]
: *NIST Special Publication 800-56A: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)*. March 2007 URL: http://csrc.nist.gov/publications/nistpubs/800-56A/SP800-56A_Revision1_Mar08-2007.pdf

[SP800-67]
: *Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, Revised January 2012.* SP-800-67 Revision 1. U.S. Department of Commerce/National Institute of Standards and Technology. URL: http://csrc.nist.gov/publications/nistpubs/800-67-Rev1/SP-800-67-Rev1.pdf

[URI]
: T. Berners-Lee; R. Fielding; L. Masinter. *Uniform Resource Identifiers (URI): generic syntax*. January 2005. RFC 3986. URL: http://www.ietf.org/rfc/rfc3986.txt

[XML-ENCRYPTION-REQ]
: Joseph Reagle. *XML Encryption Requirements*. 4 March 2002. W3C Note. URL: http://www.w3.org/TR/2002/NOTE-xml-encryption-req-20020304

[XML-NAMES]
Richard Tobin et al. Namespaces in XML 1.0 (Third Edition). 8 December 2009. W3C Recommendation. URL: http://www.w3.org/TR/2009/REC-xml-names-20091208/
[XML10]
C. M. Sperberg-McQueen et al. Extensible Markup Language (XML) 1.0 (Fifth Edition). 26 November 2008. W3C Recommendation. URL: http://www.w3.org/TR/2008/REC-xml-20081126/
[XMLDSIG-CORE1]
D. Eastlake; J. Reagle; D. Solo; F. Hirsch; T. Roessler; K. Yiu. XML Signature Syntax and Processing Version 1.1. 11 April 2013. W3C Recommendation. URL: http://www.w3.org/TR/2013/REC-xmldsig-core1-20130411/
[XMLSCHEMA-1]
Henry S. Thompson et al. XML Schema Part 1: Structures Second Edition. 28 October 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/
[XMLSCHEMA-2]
Paul V. Biron; Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition. 28 October 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/
[XPATH]
James Clark; Steven DeRose. XML Path Language (XPath) Version 1.0. 16 November 1999. W3C Recommendation. URL: http://www.w3.org/TR/1999/REC-xpath-19991116/

B.2 Informative references

[Davis]
Defective Sign & Encrypt in S/MIME, PKCS#7, MOSS, PEM, PGP, and XML. D. Davis. USENIX Annual Technical Conference. 2001. URL: http://www.usenix.org/publications/library/proceedings/usenix01/davis.html
[ECC-ALGS]
D. McGrew; K. Igoe; M. Salter. RFC 6090: Fundamental Elliptic Curve Cryptography Algorithms. February 2011. IETF Informational RFC. URL: http://www.rfc-editor.org/rfc/rfc6090.txt
[MIME-REG]
N. Freed, J. Klensin. RFC 4289: Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures. December 2005. Best Current Practice. URL: http://www.ietf.org/rfc/rfc4289.txt
[OAEP-ATTACK]
Manger, James. A Chosen Ciphertext Attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as Standardized in PKCS #1 v2.0. URL: http://archiv.infsec.ethz.ch/education/fs08/secsem/Manger01.pdf
[RELAXNG-SCHEMA]
Information technology -- Document Schema Definition Language (DSDL) -- Part 2: Regular-grammar-based validation -- RELAX NG. ISO/IEC 19757-2:2008. URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c052348_ISO_IEC_19757-2_2008(E).zip
[RFC3218]
E. Rescorla. Preventing the Million Message Attack on Cryptographic Message Syntax (RFC 3218). January 2002. RFC. URL: http://www.rfc-editor.org/rfc/rfc3218.txt
[SHA-1-Analysis]
McDonald, C., Hawkes, P., and J. Pieprzyk. SHA-1 collisions now 2^{52} . EuroCrypt 2009 Rump session. URL: http://eurocrypt2009rump.cr.yp.to/837a0a8086fa6ca714249409ddfae43d.pdf
[SHA-1-Collisions]
X. Wang, Y.L. Yin, H. Yu. Finding Collisions in the Full SHA-1. In Shoup, V., editor, Advances in Cryptology - CRYPTO 2005, 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings, volume 3621 of LNCS, pages 17–36. Springer, 2005. URL: http://people.csail.mit.edu/yiqun/SHA1AttackProceedingVersion.pdf (also published in http://www.springerlink.com/content/26vljj3xhc28ux5m/)
[Tobin]
R. Tobin. Infoset for external entities. 2000. URL: http://lists.w3.org/Archives/Member/w3c-xml-core-wg/2000OctDec/0054 [XML Core mailing list, W3C Member Only].
[XML-C14N]
John Boyer. Canonical XML Version 1.0. 15 March 2001. W3C Recommendation. URL: http://www.w3.org/TR/2001/REC-xml-c14n-20010315
[XML-C14N11]
John Boyer; Glenn Marcy. Canonical XML Version 1.1. 2 May 2008. W3C Recommendation. URL: http://www.w3.org/TR/2008/REC-xml-c14n11-20080502/
[XML-EXC-C14N]
Donald E. Eastlake 3rd; Joseph Reagle; John Boyer. Exclusive XML Canonicalization Version 1.0. 18 July 2002. W3C Recommendation. URL: http://www.w3.org/TR/2002/REC-xml-exc-c14n-20020718/
[XML-INFOSET]
John Cowan; Richard Tobin. XML Information Set (Second Edition). 4 February 2004. W3C Recommendation. URL: http://www.w3.org/TR/2004/REC-xml-infoset-20040204/
[XML-MT]
M. Murata, S. St.Laurent, D. Kohn. XML Media Types. IETF RFC 3023. URL: http://www.ietf.org/rfc/rfc3023.txt.
[XMLBASE]
Jonathan Marsh; Richard Tobin. XML Base (Second Edition). 28 January 2009. W3C Recommendation. URL: http://www.w3.org/TR/2009/REC-xmlbase-20090128/
[XMLENC-BACKWARDS-COMP]
Tibor Jager; Kenneth G. Paterson; Juraj Somorovsky. One Bad Apple: Backwards Compatibility Attacks on State-of-the-Art Cryptography. 2013. URL: http://www.nds.ruhr-uni-bochum.de/research/publications/backwards-compatibility/
[XMLENC-CBC-ATTACK]
Tibor Jager; Juraj Somorovsky. How to Break XML Encryption. 17-21 October 2011. CCS'11, ACM. URL: http://www.nds.ruhr-uni-bochum.de/research/publications/breaking-xml-encryption/
[XMLENC-CBC-ATTACK-COUNTERMEASURES]
Juraj Somorovsky; Jörg Schwenk. Análisis técnico de contramedidas contra ataques al cifrado XML, o simplemente otra motivación para el cifrado autenticado . 2011. URL: http://www.w3.org/2008/xmlsec/papers/xmlEncCountermeasuresW3C.pdf
[XMLENC-CORE1-CHGS]
Federico Hirsch. Explicación funcional de XML Encryption 1.1 . 11 de abril de 2013. Nota del grupo de trabajo del W3C. URL: http://www.w3.org/TR/2013/NOTE-xmlenc-core1-explain-20130411/
[XMLENC-DECRIPT]
Takeshi Imamura; Merlín Hughes; Hiroshi Maruyama. Transformación de descifrado para firma XML . 10 de diciembre de 2002. Recomendación del W3C. URL: http://www.w3.org/TR/2002/REC-xmlenc-decrypt-20021210

[XMLENC-PKCS15-ATAQUE]
Tibor Jäger; Sebastián Schinzel; Juraj Somorovsky. *El ataque de Bleichenbacher ataca de nuevo: romper PKCS#1.5 en cifrado XML* . 2012. URL: http://www.nds.rub.de/research/publications/breaking-xml-encryption-pkcs15.pdf

[XMLSEC-RELAJARSE]
Makoto Murata; Federico Hirsch. *Seguridad XML Esquemas RELAX NG* . 11 de abril de 2013. Nota del grupo de trabajo del W3C. URL: http://www.w3.org/TR/2013/NOTE-xmlsec-rngschema-20130411/

[XMLSEC11-REQS]
Federico Hirsch; Tomás Roessler. *Requisitos de seguridad XML 1.1 y consideraciones de diseño* . 11 de abril de 2013. Nota del grupo de trabajo del W3C. URL: http://www.w3.org/TR/2013/NOTE-xmlsec-reqs-20130411/